ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

**NEUMANN JÁNOS**
**INFORMATIKAI KAR**


# SZAKDOLGOZAT

# DIPLOMAMUNKA


| | | |
|---|---|---|
| **OE-NIK** | Hallgató neve: | **Nagy Dávid** |
| **2022** | Hallgató törzskönyvi száma: | **T/006320/FI12904/N** |

Óbudai Egyetem
Neumann János Informatikai Kar
Kiberfizikai Rendszerek Intézet

# SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Nagy Dávid**
Törzskönyvi száma: T/006320/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

**Fakataszter rendszer fejlesztése historizált NoSQL adatbázissal**
**Tree cadastre system development with historical NoSQL database**

Intézményi konzulens: Rusznák Attila
Külső konzulens:

Beadási határidő: 2022. május 15.

A záróvizsga tárgyai: Számítógép architektúrák
Big Data és üzleti intelligencia specializáció

**A feladat**

Tervezzen és valósítson meg egy fakataszter rendszert, amely lehetővé teszi, hogy egy android mobilalkalmazás segítségével fákat lehessen eltárolni egy NoSQL adatbázisban és a feltöltött fák adatait egy webalkalmazáson keresztül lehessen kezelni, historizáltan módosítani. Kutassa fel és vizsgálja meg a modern android fejlesztés architekturális komponenseit, a fakataszter rendszerek alapvető követelményeit. Ismertesse valamely NoSQL adatbázis historizációs lehetőségeit. Valósítson meg egy olyan tesztrendszert a mobil- és webalkalmazással, mely eleget tesz a fakataszter működési követelményeinek.

**A dolgozatnak tartalmaznia kell:**

- a feladat pontos ismertetését,
- a követelmény és hardware specifikációt,
- a fakataszter rendszerek tulajdonságait,
- a korszerű android fejlesztés alapelveit,
- a modern webfejlesztési technológiákat,
- a historizáció működését a választott NoSQL adatbázisban,
- a tervezés és megvalósítás részletes leírását,
- a tesztelés tervét és eredményeit,
- a továbbfejlesztési lehetőségek számba vételét.


Fleiner R

.....

Dr. Fleiner Rita
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

Rusenik Attila

.....
intézményi konzulens

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022.05.06.

.....
hallgató aláírása

**KONZULTÁCIÓS NAPLÓ**

Hallgató neve: NAGY DÁVID | Neptun kód: [redacted] | Tagozat: NAPPALI

Telefon: [redacted] | Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

FAKATASZTER RENDSZER FEJLESZTÉSE HISTORIZÁLT NOSQL ADATBÁZISSAL

~~Szakdolgozat~~ / Diplomamunka[2] címe angolul:

TREE CADASTRE SYSTEM DEVELOPMENT WITH HISTORICAL NOSQL DATABASE

Intézményi konzulens: RUSZNÁK ATTILA | Külső konzulens: .....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

| Alk. | Dátum | Tartalom | Aláírás |
|---|---|---|---|
| 1. | 2021.09.20 | A szakdolgozat felépítésének megbeszélése | |
| 2. | 2021.10.15 | Az irodalomkutatás egyeztetése | |
| 3. | 2021.11.17 | A formai követelmények pontosítása | |
| 4. | 2021.12.03 | Beadás előtti ellenőrzések | |

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2021.12.04.

.....
intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

## KONZULTÁCIÓS NAPLÓ

Hallgató neve: NAGY DÁVID | Neptun kód: [redacted] | Tagozat: NAPPALI

Telefon: [redacted] | Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat / Diplomamunka[5] címe magyarul:

FAKATASZTER RENDSZER FEJLESZTÉSE HISTORIZÁLT NOSQL ADATBÁZISSAL

~~Szakdolgozat~~ / Diplomamunka[6] címe angolul:

TREE CADASTRE SYSTEM DEVELOPMENT WITH HISTORICAL NOSQL DATABASE

Intézményi konzulens: RUSZNÁK ATTILA | Külső konzulens: .....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

| Alk. | Dátum | Tartalom | Aláírás |
|---|---|---|---|
| 1. | 2022.02.10 | Rendszerterv megbeszélése | |
| 2. | 2022.03.11 | Fejlesztési kérdései megvitatása | |
| 3. | 2022.04.08 | Tesztelés megbeszélése | |
| 4. | 2022.04.22 | Formai követelmények véglegesítése | |

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4[7] tantárgy követelményét teljesítette, beszámolóra / védésre[8] bocsátható.

Budapest, 2022.04.29.....

.....
intézményi konzulens

[5] Megfelelő aláhúzandó!
[6] Megfelelő aláhúzandó!
[7] Megfelelő aláhúzandó!
[8] Megfelelő aláhúzandó!

# Tartalomjegyzék

# 1. Bevezetés

Szakdolgozatom témáját egy természetvédelmi egyesület által igényelt fakataszter rendszer adta. Az egyesület Nyíregyházán működik és azzal bízták meg cégemet, hogy fejlesszünk számukra egy olyan rendszert, amely képes eltárolni a településen található fák szöveges és képi adatait historizáltan. Szükségük van egy mobilalkalmazásra, amely segítségével az egyesület tagjai képesek feltölteni egy fát az adatbázisba, ezentúl egy böngészőn elérhető adminisztrációs rendszerre a vezetőségnek, amely képes ezeket az adatokat megjeleníteni.

A mobilalkalmazást tekintve azt kérte az egyesület, hogy android operációs rendszeren lehessen futtatni és csak az autentikált felhasználók tudják használni, hiszen egyelőre csak a tagoknak kell hozzáférni. Ahhoz, hogy a mobil applikáció tudjon kommunikálni közvetett módon az adatbázissal szükség lesz egy REST API alkalmazásra, amely végpontjait a mobil és webalkalmazás fogja használni. Az ehhez szükséges szerver a cég számára már rendelkezésre áll, így ezzel kapcsolatban már nem kell érdemi munkát végezni. Az adatbázissal kapcsolatban az egyetlen ügyfél igény az, hogy legyen verziókövetés, tehát ha valamely fának módosítják az adatait, akkor visszakövethető legyen, hogy korábban milyen tulajdonságai voltak a fának. Így az adatbázis tervezésekor figyelembe kell venni, hogy szükség lesz historizációra.

Jelenleg a projekt még nincs elkezdve, tehát full stack fejlesztést kell majd folytatnom, amely magában foglalja a frontend kódot a web és mobil oldalon, illetve a backend kódot a REST API alkalmazásnál. Így utána kell néznem, hogy jelenleg milyen trendek vannak a mobil fejlesztés területén, milyen architekturális tervezési mintákat használnak és ezek közül melyek lesznek számomra hasznosak. Hasonlóan, ugyanígy fel kell kutatnom a legelterjedtebb webes frontend könyvtárakat, keretrendszereket, amelyek segítségével egy kész, „production-ready” alkalmazást lehet fejleszteni.

A fejlesztés során az egyesületből rendelkezésre áll majd egy tag, aki az általános fakataszter rendszer követelményeit, tulajdonságait fogja velem ismertetni, hogy az elkészült rendszer mindenképp megfeleljen számukra és hosszútávon használható legyen.

A szakdolgozatom során kezdetben megismerem az általános fakataszterek működését, követelményeit, tulajdonságait. Majd ezen ismeretek tekintetében elkezdem a mobilalkalmazás fejlesztését, hogy az ügyfél minél hamarabb lásson érdemi eredményt a munkámból. A mobilalkalmazás fejlesztésének befejezése után pedig kiválasztom a megfelelő adatbázist és elkezdem a REST API alkalmazás fejlesztését, amelyhez párhuzamosan készítem az adminisztrátor számára a webalkalmazást is. Végül tesztelem az elkészült fakataszter rendszert.

# 2. Fakataszter

## 2.1. Fakataszter jelentősége

A zöldterületek a települések nélkülözhetetlen elemei. Előnyösen befolyásolják az épített környezet ökológiai viszonyait, kedvezően hatnak a település mikroklímájára, csökkentik a hőmérsékleti szélsőségek kialakulásának veszélyét, a szennyező anyagok és a por kiszűrésében is jelentős szerepet játszanak. A zöldfelületen belül a fák a legmeghatározóbb és legértékesebb elemek.

A zöldterületekre ugyanúgy kell tekinteni, mint bármely más, a mindennapokban is vagyontárgyként kezelt elemre, például egy ingatlanra. Egy adott ingatlan is csak akkor lesz hosszú évek, évtizedek múlva értékes, ha gondját viseljük, ha foglalkozunk az állapotával és folyamatosan törekszünk arra, hogy elkerüljük a teljes amortizációt.

A fákkal ugyanez a helyzet, folyamatos karbantartást igényelnek, hogy értékesek maradjanak, sőt az évek múlásával még értékesebbek is lehessenek. A városi fák életfeltételeit a helyükön kell biztosítanunk. Ha fát ültetünk gondoskodnunk kell róla, hogy megfelelő helyre, megfelelő feltételekkel ültessük, figyelembe kell venni a talajadottságokat, talajcserét kell biztosítani. A fafenntartás leglényegesebb része a tápanyagpótlás, hiszen ettől lesz megfelelő a vitalitása. Amennyiben nem megfelelő a víz és a tápanyagellátás sokkal nehezebben reagálnak egy esetleges sérülésre, betegségre. A vegyi behatások, a sózások is könnyen tönkretehetik az állapotát, ezek minimálisra csökkentésére külön figyelmet kell fordítanunk.

A fakataszter révén lehetőségünk nyílik a fák fajtájának, méretének, egészségi állapotának és értékének meghatározására, ami igazán hasznára lehet a faállomány tulajdonosának. A fakataszter azonban akkor fog csak valódi értékkel bírni, ha hosszú távon a további munkálatok alapját képezheti. A benne lévő adatok folyamatos frissítésével megtudhatjuk, hogy az állomány hány százaléka igényel beavatkozást, szükség van-e metszésre, indokolt lehet-e egy adott fa cseréje. Az adatok naprakészen tartásával, a további cselekvési terv meghatározásával, egy hosszú távú stratégiai terv kialakításával a leghasznosabb alap-adattárként szolgálhat.

## 2.2. Fakataszter fogalma, tulajdonságai

A fakataszter tulajdonképpen egy leltár. Fő feladata, hogy bizonyos területen lévő fákat képes legyen nyilvántartani kertészeti, szakmai szempontból. Ezzel elősegítve a fenntartási időrendi meghatározását, technológiai tervezhetőségét. Fontos, hogy a fakataszter felállításának módját, technológiai leírását és kötelező adattartalmát jelenleg nem szabályozza jogszabály.

Mivel egy adott területen a faállomány folyamatosan változik, a fakataszter pontossága nem lehet százszázalékos, tökéletes pontosságú.

A kataszterek elkészítéséhez különböző egyesületek, változó útmutatókat adtak ki. Ezek nem minden pontban egyeznek, így még az egyes rendszerek között is előfordulhat jelentős különbség. A különböző egyesületek, mint például a „Magyar Faápolók Egyesülete” által készített útmutatók nem lesznek vizsgálva, hanem az ügyfél (szakértő) által megfogalmazott igények mentén lesz kialakítva a rendszer.

### 2.3. Fakataszter adattartalma

A fakataszter egy kertészeti szakmai nyilvántartási adathalmaz. Esetünkben fákat és kivágott, eltávolított fákat, azaz fatuskókat kíván tárolni a megrendelő. Tehát elmondhatjuk, hogy két fő típusú entitásról fogunk különböző információkat tárolni.

A fakataszter nem tartalmazza a kertészeti gyakorlatban cserjeként számontartott, cserjeként ültetett, vagy technológiailag cserjeként fenntartott fásszárú növényeket, illetve az egybefüggő zöldterületet (pl.: gyep, fű).

Gyakori, hogy külön jelölést kapnak a fasorok. Fasornak nevezzük az úttest mellett álló egyedi, vagy zöldsávos fahelyben elhelyezkedő fákat, amelyet a parktól szilárd burkolatú járda választ el.

A fakataszterben tárolt fák folyamatosan változnak. Léteznek olyan fakataszterek, amelyek nem historizáltak, így az adatbázisban a fák adatainak legutolsó verzióját tárolják. A folyamatos felülírás miatt az adatok változása nem nyomon követhető, ez a kevésbé értékes, hosszú távon nem igazán használható rendszer. Ezzel szemben egyre több fakataszter alkalmazza a historizálást, ami javítja a korábbi hibákat, ezentúl részletesebb és alaposabb, jobban áttekinthető rendszert tár a felhasználó elé. Továbbá a modern rendszerek nem csak szöveges, hanem vizuális információt is tárolnak a fákról. Ilyen vizuális információ lehet a kép, amelyből akár többet is készíthetünk egy adott fához, illetve a képekhez is bevezethetünk historizálást, amely tovább fokozza az adatok értékét, ezzel együtt a mennyiségét is.

# 3. Követelmény specifikáció

## 3.1. Ügyfél

Az megrendelő egy természetvédő egyesület, amelyben vannak tagok és van egy vezető. A fakataszter specifikációját az ügyfél adja meg, aki esetünkben közvetlenül az egyesület vezetője. Az vezető egyben szakértője a fakataszter rendszereknek, így a továbbiakban fejlesztendő rendszer az ő igényeihez igazodik és az általa megfogalmazott funkciókat szükséges megvalósítani.

## 3.2. Áttekintés

A fakataszter rendszer célja, hogy az egyesület tagjai egyszerűen, gyorsan a fa tényleges helyszínén tárolhassák el a fa szöveges és képi adatait a rendszerben. Ezeket az adatokat az egyesület vezetőjének szükséges részletesen megtekintenie, felülvizsgálnia, adott esetben törölni, módosítani. Továbbá a vezető veszi fel a tagokat az egyesületbe, ezzel együtt a fakataszter rendszerbe is.

## 3.3. Felhasználói követelmények

A fakataszter rendszerben két szerepkört különböztetünk meg. Vannak a „Tagok”, akik a fák és fatuskók feltöltését végzik el és van a „Vezető”, aki az eltárolt fákat felülvizsgálja, megtekinti és módosítja.

Mindkét szerepkör követelményeit az egyesület vezetőjével együtt határoztam meg. Számukra egy olyan fakataszter rendszerre van szükség, amely az adatokat historikusan tárolja a hozzá tartozó képekkel együtt. A fatuskó esetében csak szöveges információt tárolunk nem historikusan. Ezzel szemben a fa esetében mind a szöveges, mind a képállományokat historikusan kell kezelni.

Ahhoz, hogy a tagok egyszerűen, gyorsan, a fa helyszínén tárolhassák el a fával kapcsolatos szöveges és képi adatokat egy olyan platformot kellett választani, ami erre képes, így a választás az ügyfél részéről a mobil környezetre esett, ami esetünkben valóban teljes mértékben megfelelő. Ennek megfelelően a tagok mobilalkalmazást fognak használni.

A vezető pedig az eltárolt fákat, fatuskókat és rengeteg adatait főleg asztali számítógépen vagy laptopon tekinti meg. Előfordulhat, hogy több operációs rendszert használ, így az asztali alkalmazás számára nem a legjobb választás. Végül abban állapodtunk meg, hogy egy webalkalmazás lesz a helyes választás, hiszen a megnyitásához csak egy böngészőre van szükség és akár mobilról is beléphet a rendszerbe. Így a vezető számára az adminisztrációs rendszer egy webalkalmazás lesz.

### 3.3.1. Követelmények – Tag – Mobilalkalmazás

- Képes bejelentkezni az alkalmazásba a vezető által kapott adatokkal
- Képes kijelentkezni a rendszerből
- Képes megtekinteni a rendszerben eltárolt fatuskókat térkép segítségével
    - Képes a térképet műhold stílusban használni
    - Képes a térképet alapértelmezett stílusban használni
- Képes megtekinteni a rendszerben eltárolt fákat térkép segítségével
    - Képes a térképet műhold stílusban használni
    - Képes a térképet alapértelmezett stílusban használni
- Képes új fatuskó rögzítésére
    - Ha a GPS be van kapcsolva, a térkép azonnal a felhasználó pozíciójára közelít
    - Képes térképen bejelölni egy marker segítségével az új fatuskó pozícióját
- Képes új fa rögzítésére
    - Ha a GPS be van kapcsolva, a térkép azonnal a felhasználó pozíciójára közelít
    - Képes térképen bejelölni egy marker segítségével az új fa pozícióját.
        - A rendszer az vizsgálja meg az elhelyezett marker pozícióját és lehetőleg töltse ki a lokációval kapcsolatos beviteli mezők értékét. Ezen mezőket a felhasználó is módosíthatja, amennyiben a rendszer nem megfelelő adatokkal töltötte ki a beviteli mezőket.
    - Képes az új fáról 3 képet készíteni, amelyeket szintén tárol a rendszer
        - A képeket a rendszer tömörítse az adatforgalom csökkentése érdekében
- Képes új hasonló fa rögzítésére, ekkor a legutóbb hozzáadott fa bizonyos adatait nem szükséges újra beírni. Ez megkönnyíti egy fasor felvitelének sebességét.

Igényeik:

- Az adatfelvitel egyszerű, felhasználóbarát megvalósítása
- Beviteli mezők ellenőrzése a hiányos adatfelvitel megelőzése érdekében
- Az átláthatóság kedvéért külön menüpont legyen a fatuskó és a fa hozzáadása
- A fa hozzáadása 3 lapon történjen:
    - Lokáció
    - Fa tulajdonságok
    - Képek hozzáadása

### 3.3.2. Követelmények – Vezető – Webalkalmazás

- Képes bejelentkezni az adminisztrációs rendszerbe
- Képes kijelentkezni az adminisztrációs rendszerből
- Képes megtekinteni a fatuskókat táblázatos nézetben
  - Képes szűrni a fatuskók között
- Képes megtekinteni a fákat táblázatos nézetben
  - Képes szűrni a fák között
- Képes megtekinteni egy fának a részletes szöveges és képi adatait
  - Képes megtekinteni a fát feltöltő tagot és a tag adatait
- Képes új „Tag” szintű felhasználót regisztrálni a rendszerbe, aki a mobilalkalmazást tudja használni
  - A regisztrált tag az e-mail címére kap egy üzenetet, amely a belépési adatokat tartalmazza
- Képes megtekinteni a regisztrált tagokat
- Képes inaktiválni a tagokat, hogy ideiglenesen ne tudjanak bejelentkezni a mobilalkalmazásba
- Képes törölni a regisztrált tagokat
- Képes megtekinteni egy adott fa korábbi verziójához tartozó adatait

Igények:

- Egyszerű, gyors szűrés és keresés a fák és fatuskók között
- Verziókövetés megvalósítása a fák esetében, módosítás során
- Képek megjelenítése az egyes fákról
- Tagok felülvizsgált regisztrálása a rendszerbe

## 3.4. Funkcionális követelmények

### 3.4.1. Autentikáció

A mobilalkalmazásba a tagot a vezető regisztrálja. Az egyesület tagjainak a vezető bemutatót tart a rendszer használatáról, tulajdonságairól. Miután minden tag megismerte a zárt megbeszélésen ismertetett rendszert, a vezető belép az adminisztrációs rendszerbe és regisztrálja a tagokat. A tagok adatai között szerepelnek a hitelesítési adatok, mellyel a tagok a mobilalkalmazásba be tudnak lépni. Ilyen az e-mail és a hozzá tartozó jelszó. Regisztrációt követően ezek az adatok nem módosíthatók.

Az e-mail és jelszó párossal tudja a tag használni a mobilalkalmazást, amely csak addig tárolja a bejelentkezési adatokat, amíg a mobilalkalmazás aktívan vagy a háttérben fut. Minden egyes bejelentkezésnél vizsgálja a rendszer, hogy a tag nincs-e inaktiválva vagy törölve.

### 3.4.2. Fa feltöltés folyamata

Ahhoz, hogy egy fát minél egyszerűbben, gyorsabban lehessen feltölteni, a szükséges adatok bevitelét kell egyszerűsíteni azzal, hogy minél kevesebbet kelljen gépelni a mobilalkalmazásban. Emiatt a lehető legtöbb beviteli mező nem kézzel írt szöveget vár, hanem egy előre definiált listát nyújt a tagnak, akinek csak ki kell választania a megfelelő jellemzőt.

A lokációval kapcsolatos adatok megadásakor egy térképen is jelezni kell egy marker segítségével az újonnan elhelyezendő fát. Annak érdekében, hogy minél kevesebbet kelljen gépelni, a marker elhelyezésekor a rendszer automatikusan kitölti a város, utca és házszám mezőket. Ezek a mezők kézzel is módosíthatóak, hiszen nem biztos, hogy minden esetben képes a megfelelő címet felismerni a rendszer.

A képek feltöltése lapon fontos, hogy a tag lássa az elkészített képet, így a fotó után megtekinthető egy előnézet mindhárom képből. Továbbá a képek feltöltése tömörítés nélkül rengeteg adatforgalmat generál, így a küldés gombra kattintáskor a képeknek egy tömörített verziója kerül fel a fakataszterbe.

# 4. Korszerű Android alkalmazás fejlesztés

## 4.1. Android operációs rendszer

Az Android egy Linux alapú operációs rendszer, amely elsősorban a mobil készülékek és tabletek részére készült. A rendszer kifejlesztésének hátterében a Google áll. Az Android operációs rendszer lehetővé teszi, hogy a fejlesztők Java vagy Kotlin nyelvben írják meg a különböző applikációkat. A 2008 –as évben megjelenő legelső változat óta már rengeteg év telt el, de az Android továbbra is az egyik legkedveltebb mobil applikációnak számít.

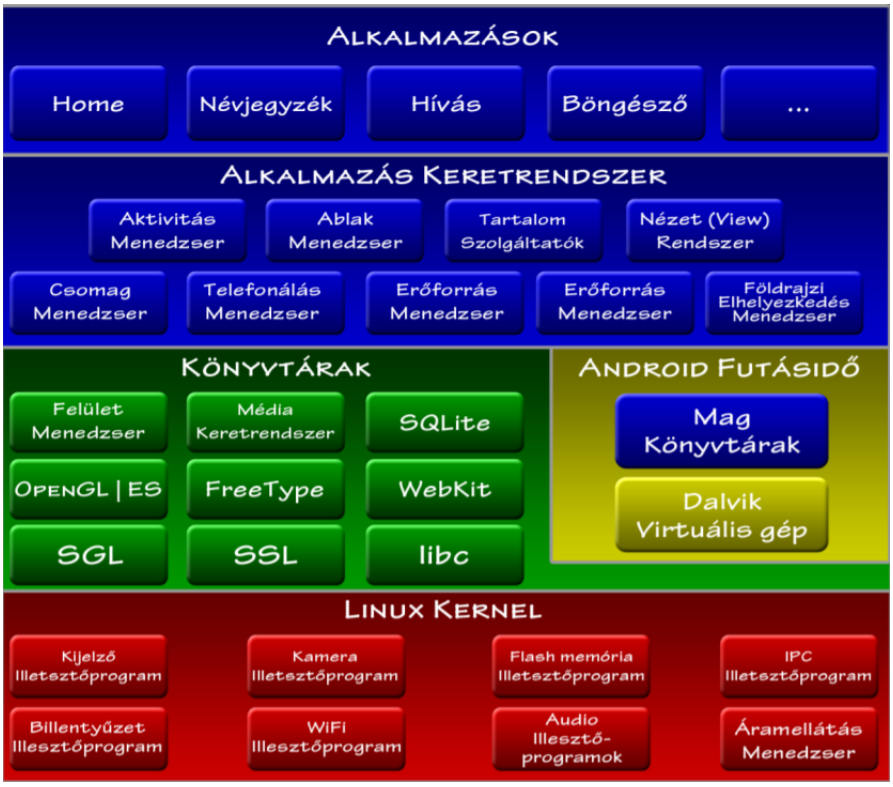
Az Android SDK olyan eszközöket és alkalmazásprogramozási felületeket (APIs) biztosít, melyek szükségesek az Android platformon való alkalmazásfejlesztés elkezdéséhez, mindezt Java vagy Kotlin programozási nyelven.

## 4.2. Android architektúra

Az Android architektúra alapját a Linux kernel adja, amely a hardver által kezelendő eszközök meghajtóprogramjait tartalmazza. Továbbá ez a kernel adja a memória kezelését, a folyamatok ütemezését és az alacsony fogyasztást elősegítő teljesítménykezelést is.

A kernel szolgáltatásait a Linux rendszerekben meglévő különféle programkönyvtárak használják, mint például az SQLite, WebKit, SSL. Részben épül ezekre az Android Runtime (ART) és a Dalvik virtuális gép. A programok nem .class állományba kerülnek fordítás után, hanem egy Dalvik Executable (.dex) formátumba. Ez általában kisebb, mint a forrásul szolgáló .class állományok mérete.

Ezután található az architektúra legfelső szintje, amely az alkalmazás keretrendszert és az alapvető alkalmazásokat (böngésző, névjegyzék) foglalja magába.




**1. ábra: Android architektúra**

### 4.3. Alkalmazás komponensek

Az Android alkalmazások fejlesztése során különböző komponensekből építjük fel az alkalmazásunkat. A komponensek az alkalmazások építőkövei. Ezek a komponensek valamilyen módon kapcsolatban lehetnek egymással, de akár teljes mértékben különállóan is működhetnek, így külön entitásként is tekinthetünk rájuk. A komponenseken keresztül a rendszer elérheti az alkalmazást.

Összesen négy különböző típusát különböztetjük meg a komponenseknek. Minden komponenstípus különböző célt szolgál, valamint különböző életciklussal rendelkeznek, ami meghatározza, hogy hogyan jön létre és áll le a komponens.

1). **Activity:** Valójában egy felhasználói felületet reprezentál. Az activity az Android appok kulcsfontosságú komponense. Az egyik legalapvetőbb belépési pontja egy alkalmazásnak. Egy alkalmazásnak lehet több activiy-je és ezek egymástól akár teljesen függetlenül is működhetnek.

2). **Service:** A service egy olyan komponens, amely segítségével úgy érhetjük el az alkalmazásunkat, hogy nem szükséges ahhoz felhasználói interakció. Kifejezetten háttérben futó hosszabb folyamatok futtatására használják, amelyek nem igénylik a felhasználói beavatkozást.

3). **Content provider:** A content provider kezeli a hozzáférést egy központi adattárhoz. A provider az Android alkalmazás része, amely elsősorban más alkalmazások számára biztosít hozzáférést egy kliens objektumon keresztül. Általában a következő két esetben dolgozunk content provider-rel. Az egyik az, amikor egy másik app content provider-jéhez akarunk kapcsolódni, amely abból az alkalmazásból valamilyen adatot szolgáltat számunkra. A másik lehetőség pedig az, hogy az alkalmazásunkban el kell készíteni egy saját content providert, amelyet azért készítjük el, hogy majd egy külsős alkalmazás használja a jövőben.

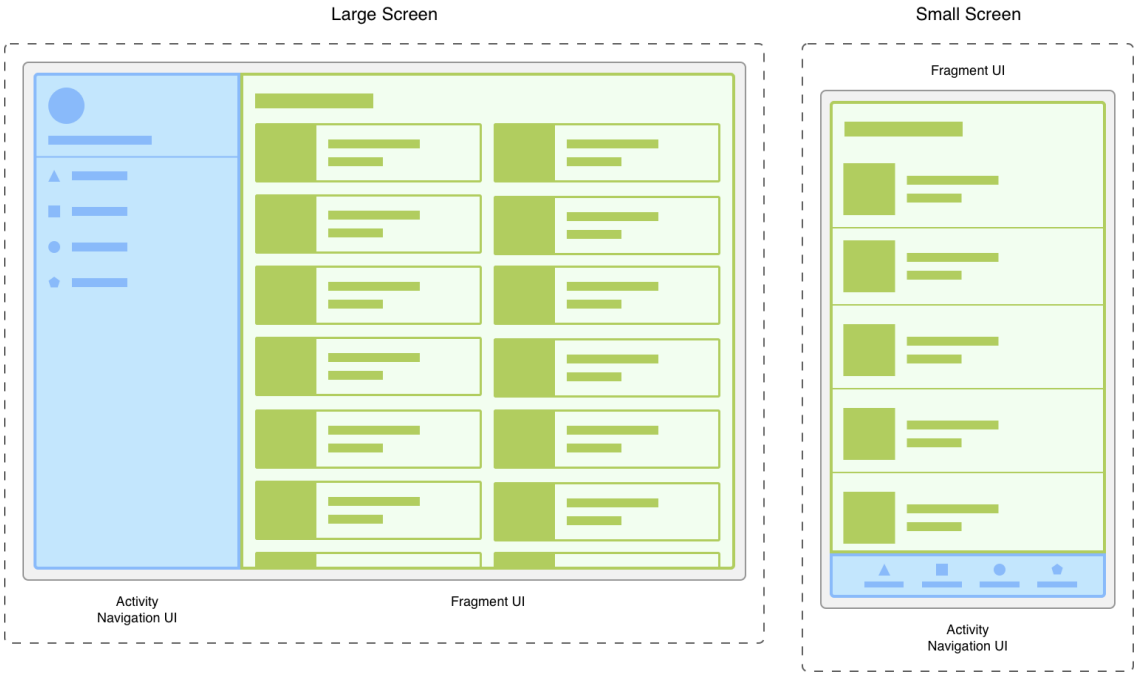
4). **Broadcast receiver:** Az Android alkalmazások „broadcast" üzeneteket küldhetnek és fogadhatnak az Android rendszerből és másik Android appoktól hasonlóan a „publish-subscribe" (közzétesz-feliratkoz) tervezési mintához. Az üzeneteket valamilyen esemény hatására lehet kiküldeni. Ilyen például, amikor történik valamilyen rendszeresemény: rendszerindítás vagy az eszközt elkezdi feltölteni a felhasználó. Ezentúl az appok egyéni üzeneteket is „sugározhatnak", hogy más alkalmazásokat értesítsenek valamiről.

## 4.4. Korszerű Android alkalmazás felépítése

### 4.4.1. Fragment

Az Android alkalmazásunkat többféle felépítésben fejleszthetjük le. Az alapvető megközelítés az lenne, hogy minden egyes képernyő az alkalmazásunkban egy új activity, viszont ezzel az a probléma, hogy az activity egy alkalmazás komponens, amelyet nehéz karbantartani és létrehozása rengeteg erőforrást igényel, így végső soron egy optimalizálatlan alkalmazást kapunk. A Google a „több activity" megközelítés helyettesítésére létrehozta a fragment-et, amely az activity moduláris részeként funkcionál.

Ennek köszönhetően napjainkban már az „egy activity, több fragment" tervezést tekintik a modern megközelítésnek, hiszen ez azzal jár, hogy a kevés erőforrásigényes fragment cseréjére van csak szükség a különböző oldalak váltásakor és nem kell egy teljesen új activity-t létrehozni.

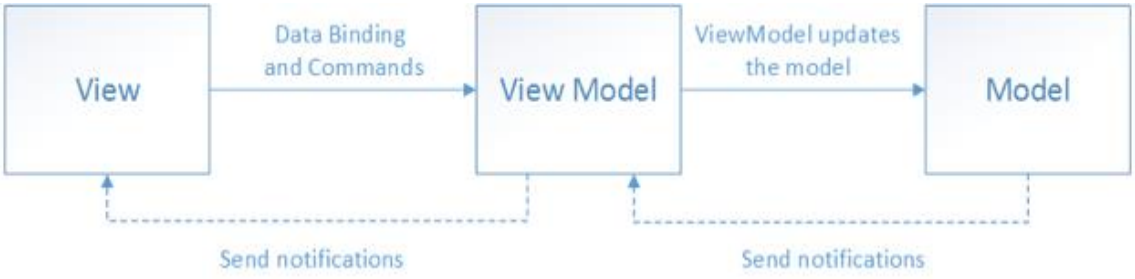



**2. ábra: Fragment és Activity**

### 4.4.2. MVVM

Alkalmazás fejlesztés során a megfelelő, karbantartható és újrahasznosítható kódszerkezet létrehozásához ajánlott valamilyen architekturális mintát követni és ez alapján felépíteni az applikációnkat. Több fajta architekturális minta létezik, amelyek közül választhatunk a kód felépítésének tervezésekor, mint például az MVP (Model-View-Presenter), MVC (Model-View-Controller) vagy az MVVM (Model-View-ViewModel). A minta kiválasztásakor figyelembe kell venni azt, hogy az alkalmazást milyen platformra fejlesztjük, milyen funkciókat kell megvalósítani és akár a fejlesztői környezet vagy nyelv is befolyásolhatja a megfelelő architekturális minta kiválasztását.

Android fejlesztés esetében a kiemelten támogatott minta az MVVM, ami a modell-nézet-nézetmodell, azaz a Model-View-ViewModel. A mintában alapvetően a modell az adatokat reprezentálja, a nézetmodell a valósítja meg az üzleti logikát és a modellen hajtja végre a műveleteket, ha szükséges. Továbbá a nézet az, amely felelős a megjelenésért és használja a nézetmodellt.



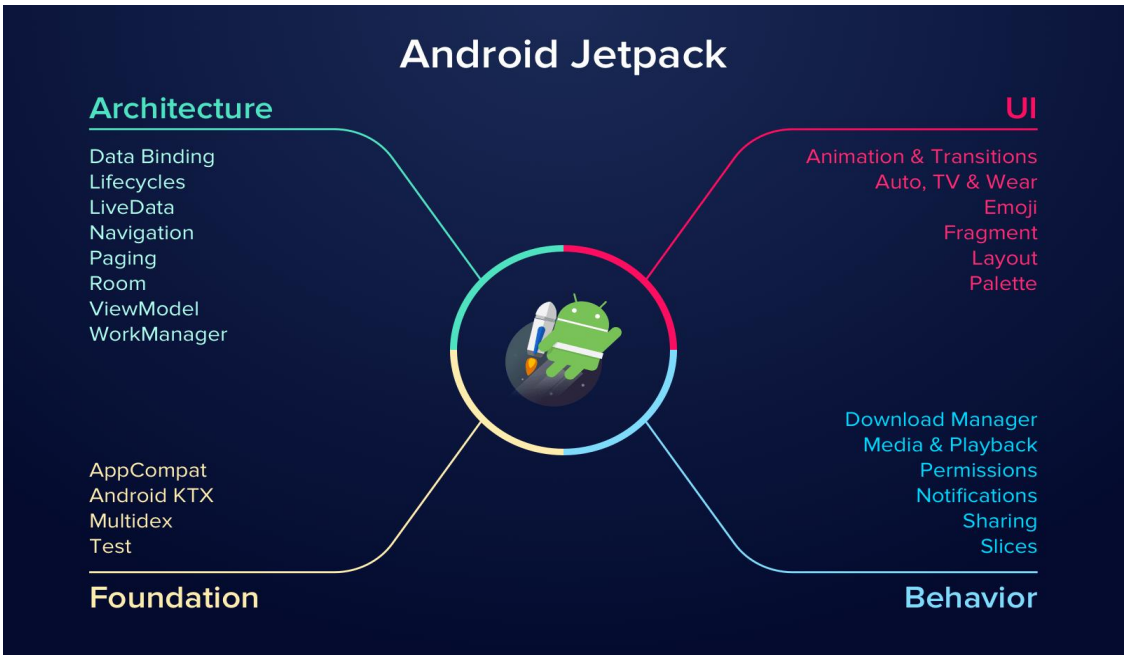

**3. ábra: MVVM**

Magas szinten a nézet „tud” a nézetmodellről és a nézetmodell szintén „tud” a modellről, de a modell nem „tud” a nézetmodellről és a nézetmodell sem „tud” a nézetről.

### 4.4.3. Android Jetpack

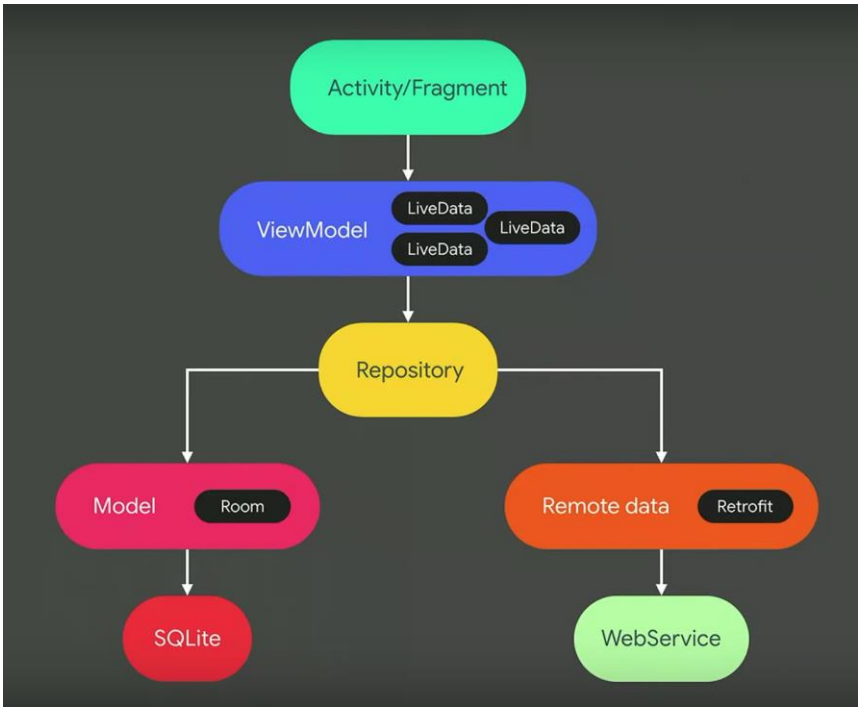
Az Android Jetpack egy olyan könyvtárcsomag, amely segít a fejlesztőknek követni a jól bevált gyakorlatokat, csökkenteni a „boilerplate” kódot és olyan kódot írni, amely konzisztensen működik minden Android verzión és eszközön, így a fejlesztő csak a saját kódja megírására összpontosíthat. A Jetpack könyvtárcsomag 2018-ban jelent meg és azóta is folyamatosan bővítik, illetve karbantartják.

Ezen könyvtárcsomagban található könyvtárak 4 kategóriára bonthatók. Ilyen kategória az „Architecture”, amelyben az architektúrával kapcsolatos komponensek, könyvtárak találhatóak, mint például a Lifecycle, ami az app életciklusát kezeli, a Data Binding, ami az adatkötésért felel, illetve a Room, amely a lokális SQLite adatbázist kezeli. A második kategória a „Foundation”, amely a fő rendszer komponenseket foglalja magába, mint például a Kotlin nyelv támogatását. A harmadik kategória a „UI”, amely a felhasználói felületért felel. Ide tartoznak az animációk, különböző layout elemek, illetve a fragmentek. Az utolsó kategória a „Behavior”, amelybe tartozik például az a könyvtár, amely az értesítéseket kezeli vagy az, amely a jogosultságokat.




**4. ábra: Android Jetpack**

Figyelembe véve az Android Jetpack nyújtotta lehetőségeket, az Android alkalmazások alapvető felépítése az, hogy a képernyőket activity-vel illetve, ha szükséges, akkor minél több fragmenttel alakítsuk ki. Az activity és fragment látja és használja a viewmodelt, ami pedig a modellel dolgozik. Attól függően, hogy az adatokat lokálisan vagy esetleg egy külső kiszolgáló során érjük el, az annak megfelelő konyvtárat használjuk.




**5. ábra: Android alkalmazás felépítés**

# 5. Webfejlesztési technológiák

## 5.1. Keretrendszer

A keretrendszer valójában az alkalmazás fejlesztését szolgáló szabályok gyűjteménye. A szabályok sokfélék lehetnek, de változtathatóak. Vannak olyan keretrendszerek, amelyet valamilyen fejlesztői közösség dolgozott ki, a saját igényüknek megfelelően, de valójában egy fejlesztő is kidolgozhat saját magának egy keretrendszert.

Az egyes szabályok tipikus problémákra adnak jól bevált megoldásokat. Ilyen például a fájlstruktúra meghatározása, amely az egyes állományokat funkciónként eltérő könyvtárba helyezzük el. Előfordul az is, hogy a fájl neve tükrözze a tartalmát, így nagyon gyakoriak az állományok elnevezésére vonatkozó követelmények is. Továbbá az egyes keretrendszerekben szabályok vonatkoznak a belépési pontra, a nyelvi beállítások, illetve a konfigurációs állományok kezelésére is.

Ezeknek köszönhetően a keretrendszer sok mindent elfed a fejlesztő elől és csak a „kényelmes” interfészt látja majd. A fejlesztő csak a lényegre tud koncentrálni, az egyéni üzleti logikára és nem szükséges minden gyakran előforduló kisebb funkciót lekódolni, ha a keretrendszer ad arra megoldást.

Összefoglalva a keretrendszereknek rengeteg előnye van és csekély mennyiségű hátránya.

**Előnyök:**

- Csökkenti a fejlesztési időt.
- Kötelező a keretrendszer által irányított struktúrát, szabályrendszert és könyvtárszerkezetet követni.
- Szétválasztott, átláthatóbb kódot eredményez.
- A kód újrafelhasználhatóbbá válik.
- Sokkal könnyebb csapatmunkát kialakítani, mivel a különböző rétegek el vannak különítve egymástól.

**Hátrányok:**

- Minden keretrendszer a saját szabályai alapján működik, amit a fejlesztőnek el kell sajátítania
- A keretrendszer által meghatározott szabályok bizonyos esetekben korlátokat is jelenthetnek.

## 5.2. Webfejlesztési keretrendszerek

A keretrendszerek a webfejlesztés elengedhetetlen részévé váltak, mivel a webalkalmazások színvonala folyamatosan emelkedik és az igényelt alkalmazások komplexitása, összetettsége is folyamatosan nő. Továbbá teljesen ésszerűtlen újra feltalálni a kereket. Éppen ezért a világszerte több ezer fejlesztő által támogatott keretrendszerek használata tűnik a legjobb megközelítésnek a gazdag és interaktív webalkalmazások készítéséhez.

Egy dinamikus, komplexebb webalkalmazásnak van FrontEnd és BackEnd oldala. Mindkét oldal programozásánál dönthetünk különbözően a keretrendszer használatát illetően. Ha úgy döntünk, hogy keretrendszert használunk, akkor külön-külön kell kiválasztanunk a számunkra megfelelő keretrendszert mind FrontEnd és BackEnd oldalon. A legjobb választáshoz vizsgáljuk meg a lehetőségeket.

### 5.2.1. BackEnd keretrendszerek

A megfelelő BackEnd keretrendszer kiválasztása során kiemelten megfigyeltem a népszerűséget a fejlesztők körében és a három legnépszerűbb tulajdonságait vizsgáltam meg.

Az egyik legnépszerűbb az **Express.js**. Az Express egy rugalmas Node.js alapú keretrendszer, amely egy stabil és robosztus funkciókészletet biztosít API, webes és mobilalkalmazások fejlesztéséhez. Megkönnyíti a Node.js alapú webalkalmazások gyors fejlesztését. A Node.js egy „runtime”, amely segítségével JavaScriptben tudunk szerver oldali kódot futtatni. Így a FrontEnd fejlesztő is könnyen beletanulhat a szerver oldali programozásba Node.js segítségével. Előnyei közé tartozik a könnyen tanulhatóság, a nagy teljesítmény, a gyors szerveroldali programozás, könnyen megvalósítható a „routing” és a hibakeresés. Hátránya, hogy alapból nem támogat biztonsági funkciókat és a hibaüzeneteket nehéz megérteni.

A következő szintén nagyon népszerű keretrendszer a **Django**. A Django egy open-source keretrendszer, amely a Python programozási nyelven alapul és MVC architekturális mintát követ. A Django alkalmas kifinomult és funkciókban gazdag adatbázis-vezérelt webalkalmazások fejlesztésére. Számos fejlesztő a Djangot tartja az egyik legjobb BackEnd keretrendszernek. Kevesebb kódolást, nagyobb újrafelhasználhatóságot és gyorsabb fejlesztést tesz lehetővé. Minden művelethez Pythont használ és opcionális adminisztrátori felületet biztosít a „CRUD” (Create, Read, Update, Delete) műveletek megvalósításához. Előnyei közé tartozik a gazdag funkcionalitás, optimális biztonság, magas skálázhatóság, egyszerű szintaxis és a sokoldalúság. Hátránya, hogy túlságosan monolitikus, minden a Django ORM-en alapul és a teljes rendszer ismerete szükséges a munkához és megfelelő használathoz.

Egy másik nagyon népszerű BackEnd keretrendszer a **Laravel**. A Laravel egy nyílt forráskódú PHP webes keretrendszer, amely az MVC architekturális mintát követi. Egy moduláris rendszert kínál, amely dedikált függőségkezelőt használ. A Laravel szintén az egyik legjobb webes keretrendszernek tartják a fejlesztők. A Laravel számos módot kínál felhasználóinak a relációs adatbázisok elérésére, valamint az alkalmazás megfelelő karbantartására. A Laravel előnyei közé tartozik az egyszerű autentikáció kialakítása, egyszerű API készítés, jó biztonság, kiváló logolás és tesztelés, ezentúl könnyű „template engine”-nel rendelkezik. Hátránya, hogy a verziók között nem igazán van átjárás, így néhány frissítés hibához vezethez, illetve a Composer (csomagkezelő) szintén nem a legjobb eszköz.

### 5.2.2. FrontEnd keretrendszerek

A webalkalmazások rendelkeznek kliens-oldali kóddal és manapság rengetek kiváló FrontEnd keretrendszer létezik. Ezek közül szintén a három legnépszerűbb, fejlesztők által legkedveltebb keretrendszereket vizsgáltam meg.

A **React** az egyik legnépszerűbb FrontEnd keretrendszer. Röviden összefoglalva ez egy JSX szintaxist használó komponens-alapú JavaScript könyvtár, amelyet a Facebook fejlesztett ki és 2011-ben vezettek be. Később, 2013-ban vált nyílt keretrendszerré, amely egy kicsit eltér a hagyományos keretrendszer definíciójától. A React legfontosabb jellemzője egy virtuális DOM (Document Object Model), egyirányú adatkötéssel. A DOM-nak köszönhetően a Reactet rengetegen kedvelik kiváló teljesítményéért és az egyik legkönnyebben megtanulható keretrendszernek tartják. Előnyei közé tartozik, hogy gyakran kap frissítést, a virtuális DOM gyorsasága, az újrafelhasználható komponensek és a Facebook által nyújtott folyamatos támogatás. Hátránya, hogy a JSX szintaxis tanulása sok időt vesz igénybe.

Egy másik gyakran használt FrontEnd keretrendszer az **Angular**. 2016-ban jelent meg az AngularJS továbbfejlesztett kiadásaként megnövelt teljesítménnyel és rengeteg erőteljes funkcióval. TypeScript alapú, ami röviden a JavaScript nyelvet egészíti ki erősen típusos nyelvre. Az Angular kétirányú adatkötést biztosít az azonnali szinkronizáció megvalósításához a modell és a nézet között. Továbbá egy hierarchikus „dependency injection”-t támogat, amely a komponenseket tesztelhetővé, újrafelhasználhatóvá és könnyebben ellenőrizhetővé teszi, valamint segít a függőségek meghatározásában. Előnyei közé tartozik, hogy támogatja a kétirányú adatkötést, stabil közösség és a Google van mögötte, illetve támogatja még a dependency injectiont, így a kód tesztelhetőségét. Hátránya, hogy kezdőknek nehéz beletanulni és kisebb projektek esetében pedig egy feleslegesen komplex keretrendszer.

A **Vue** egy szintén elterjedt JavaScript FrontEnd keretrendszer. A Vue egy virtális DOM-ot, komponens-alapú architektúrát és kétirányú adatkötést támogat. Mindez megkönnyíti a különböző komponensek használatát, módosítását, és az adatváltozások nyomon követését, amely szinte minden olyan alkalmazásnál szükséges, ahol az adatok frissítése valós időben szükséges. Előnyei közé tartozik, hogy nem generál nagy méretű projektet, gyors és kezdőknek kifejezetten barátságos, egyszerű szintaxissal rendelkezik, ezentúl támogatja a kétirányú adatkötést. Hátránya, hogy nem igazán vannak megfelelő bővítmények, teljes új, magánszemélyek által fejlesztett és korlátozottan alkalmazható nagyobb projektekre.

### 5.2.3. Next.js: A Full-Stack keretrendszer

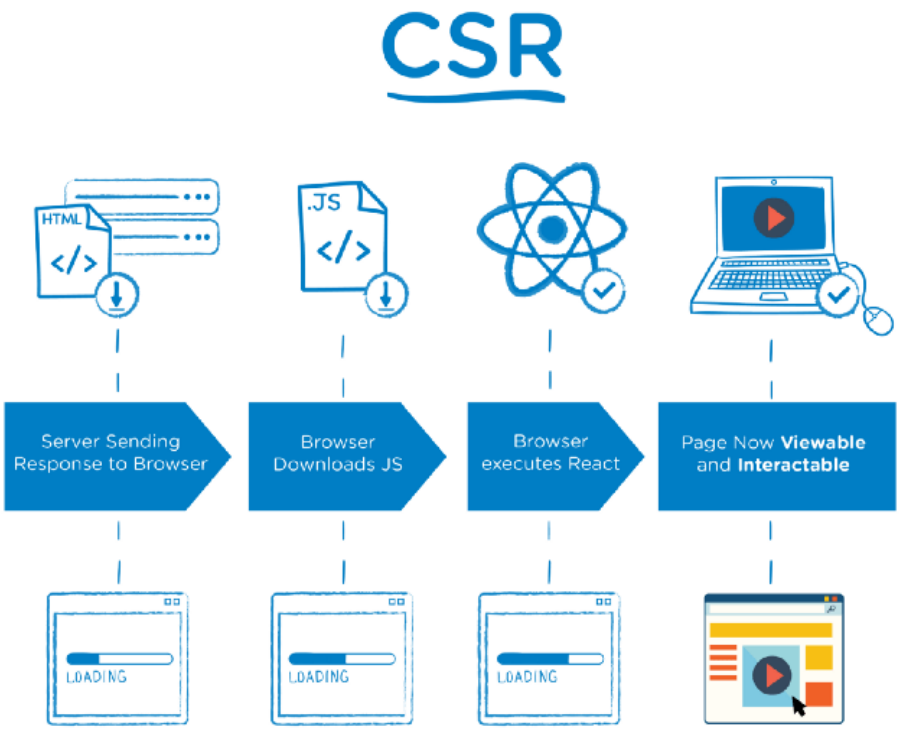
A különböző FrontEnd keretrendszereket vizsgálva arra jutottam, hogy egy olyan keretrendszert választok, amely nem csak kisebb alkalmazások fejlesztésére ajánlott, hanem egy skálázható, stabil, de semmiképp nem egy komplex keretrendszert választok a boilerplate kód és a túlvállalás elkerülése miatt. Így esett a választásom a React-re.

Ezután a számomra megfelelő BackEnd keretrendszer kiválasztása során találtam rá a Next.js-re. A **Next.js** egy Full-Stack React keretrendszer, amely olyan biztonsági és optimizációs funkciókat kínál, amely segítségével az alkalmazás sokkal hamarabb éri el a „production-ready” verziót. Olyan problémákat, feladatokat old meg, amelyeket amúgy is el kellene végeznünk, hogy ha az alkalmazásunkat ki akarjuk adni külsős felhasználóknak. Ezentúl egy Full-Stack keretrendszer, ami azt jelenti, hogy támogatja a szerver-oldali kódot Node.js-ben. Ennek köszönhetően FrontEnd és BackEnd oldalon is csak egy nyelvet, a JavaScriptet kell használnunk, viszont használhatjuk a típusos verzióját az átláthatóságért és a bugok redukálása érdekében.

A Next.js néhány hasznos funkciói és előnyei:

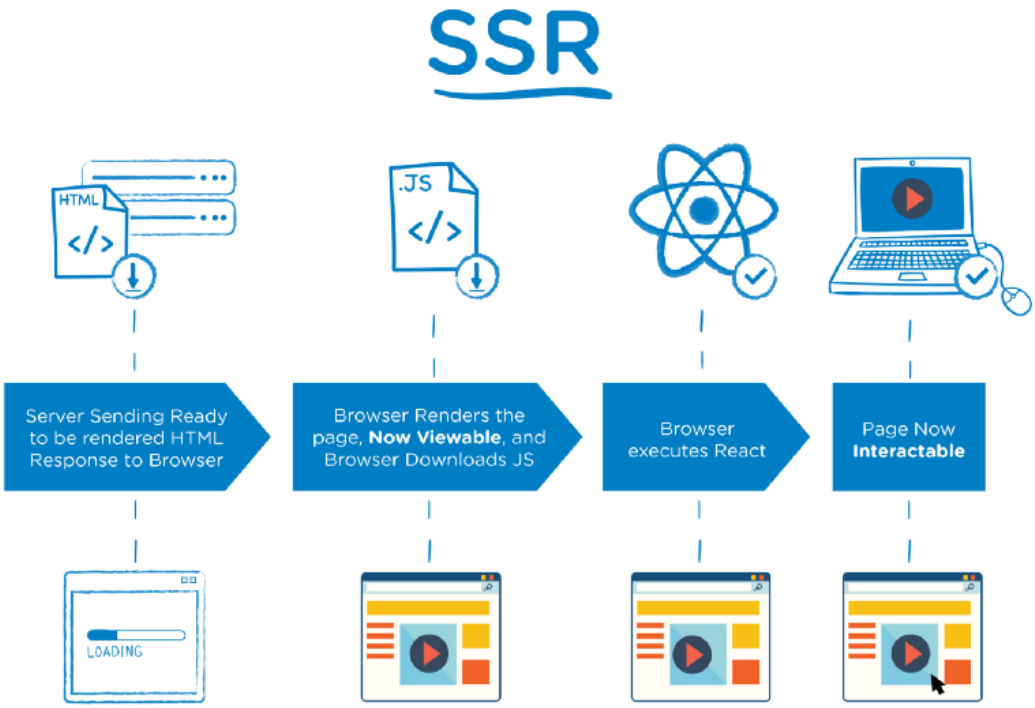
- Kép optimalizálás,
- analitika,
- szerver oldali renderelés (SSR),
- kiváló routing megvalósítás: fájlrendszer routing,
- TypeScript támogatás: típusosság miatt sokkal nehezebb hibába futni,
- zéró konfiguráció: optimalizálva van a kezdetektől fogva,
- API végpontok kialakítására is kiválóan megfelel,
- beépített CSS támogatás,
- és még számos hasznos funkcióval rendelkezik.

Számos alkalommal, amikor fejlesztünk egy modern webalkalmazást FrontEnd keretrendszerrel, egy idő után rájövünk arra, hogy nem feltétlen kellene minden JavaScript fájlt a kliens-oldalon lerenderelni, hisz ez rengeteg időt vehet igénybe a felhasználói oldalon. Minden JavaScript fájl betöltése mielőtt a felhasználói felület látható és interaktálható lenne a felhasználó számára nagyon sok időbe telhet és az egyik olyan dolog, ami miatt a webfejlesztő igazán aggódhat az az, hogy a felhasználók hogyan érzik magukat miközben használják a webalkalmazást. Ezt az élményt pedig a töltési idő nagyban ronthatja.




**6. ábra: Client Side Rendering**

A Next.js egyik leghasznosabb tulajdonsága, hogy a fejlesztő eldöntheti, hogy melyik oldal renderelése történjen a kliens és melyik a szerver oldalon.



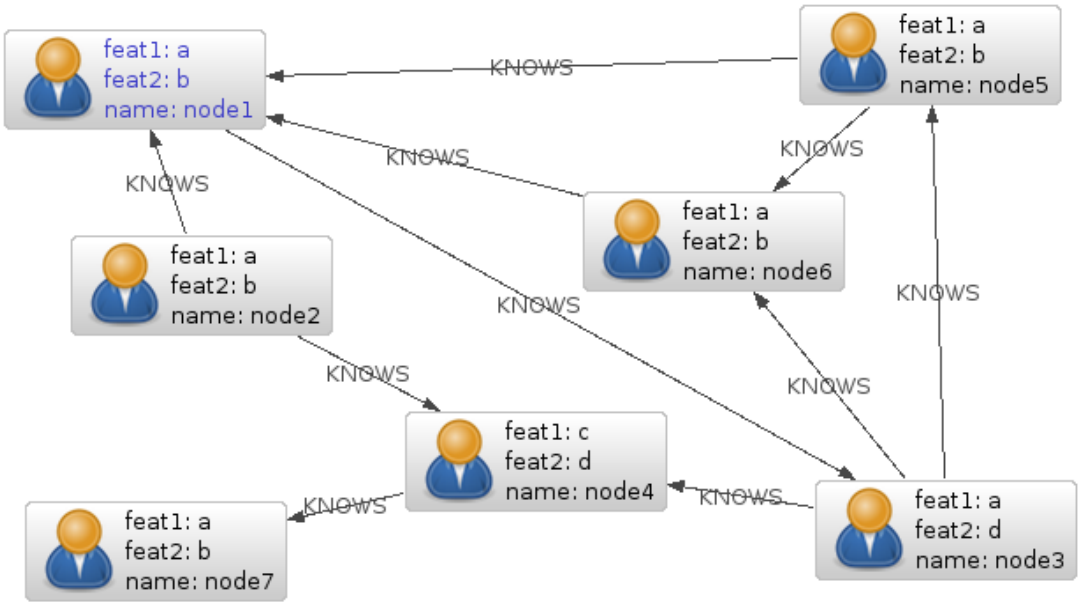

**7. ábra: Server Side Rendering**

Az alkalmazás Next.js használatával gyorsabb, mert csak a szükséges JavaScript fájlok töltődnek be minden egyes kért oldalon.

# 6. Historizáció NoSQL adatbázisokban

## 6.1. Gráf-alapú adatbázisok

A gráf adatbázisok egyre nagyobb jelentőséget kapnak, kifejezetten a közösségi oldalak tervezésekor. Például kifejezetten jól lehet ábrázolni az emberek közötti különböző kapcsolatokat vagy hierarchiát a gráf-orientált adatbázisokban, például a Neo4j-ben. Azonban a historizált adatok kezelése nem túl egyszerű módon oldható meg ilyen típusú adatbázisokban, viszont egyre több alkalmazásban követelik meg az üzleti igények a historizálást, verzió követést. A változások követése valóban az egyik fő funkciója a relációs adatbázisoknak, ugyanakkor nem szabad megfeledkeznünk a nem-relációs adatbázisokról sem.




**8. ábra: Gráf adatbázis példa**

Számos irodalom foglalkozik azzal, hogy a különböző adatbázisokat hogyan kell tervezni akkor, ha historizációt is be akarjuk építeni az alkalmazásunkba, viszont csak egy nagyon kis részük említi a historizációt NoSQL adatbázisok esetén. Éppen ezért nincs egy általános szabvány, ami alapján gráf adatbázisok esetén meg lehet valósítani a historizációt.

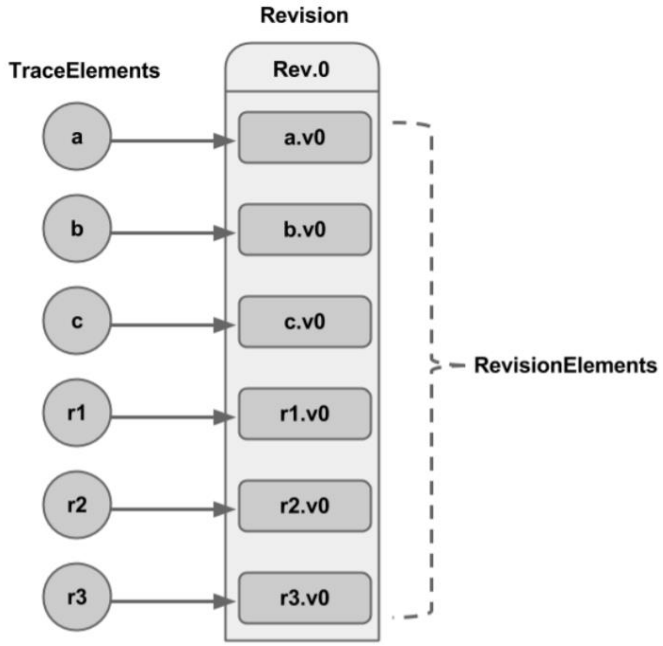
Különböző irodalmakat, publikációkat vizsgálva egy egyedi megvalósítást fogok bemutatni. Először is ez a megvalósítás a következő kritériumokat figyelembe véve jött létre:

- **Intruzív mentesség kritériuma**
  Technikai szinten a rendszernek úgy kell működnie, hogy a fejlesztőknek nem kell kódot módosítaniuk, kivéve ha valamilyen extra támogató historizációval kapcsolatos funkcionalitást kell hozzáadniuk a rendszerhez.
- **Csatlakoztathatóság kritériuma**
  A rendszert egy plugin vagy könyvtárként lehessen alkalmazni.
- **Időbeli függetlenség kritériuma**
  A rendszernek bármikor csatlakoztathatónak kell lennie.
- **History támogatás kritériuma**
  A rendszernek valamilyen módon képesnek kell lenni arra, hogy egy csomópont (node) vagy kapcsolat korábbi adatait szolgáltassa.

Ezentúl szükség van arra, hogy meghatározzuk a különböző history típusokat, amelyek megjelenhetnek egy gráf adatbázis esetében:

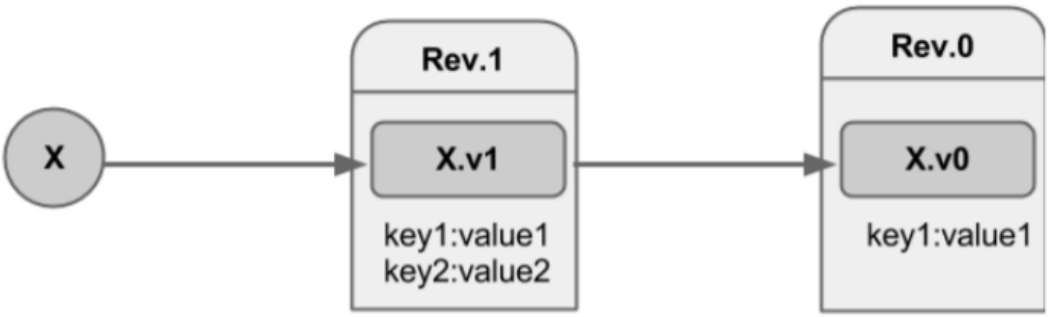
- Az első típus esetében a változás a **csomóponton**, azaz az entitáson történt.
- A második típus amikor a változás a **kapcsolaton** történt.
- A harmadik típus, amikor a változás magán a **gráfon** történt.

A jelen megközelítés azon alapszik, hogy az operatív adatok, amely a naprakész adatot tartalmazza egy teljesen más modellezést követ, mint a verzionált adat. Verzió gráfnak hívjuk azt a gráfot, amely a különböző verziók adatait, a historizált adatot tárolja. Ezt a 9. ábrán látható módon teszi meg.



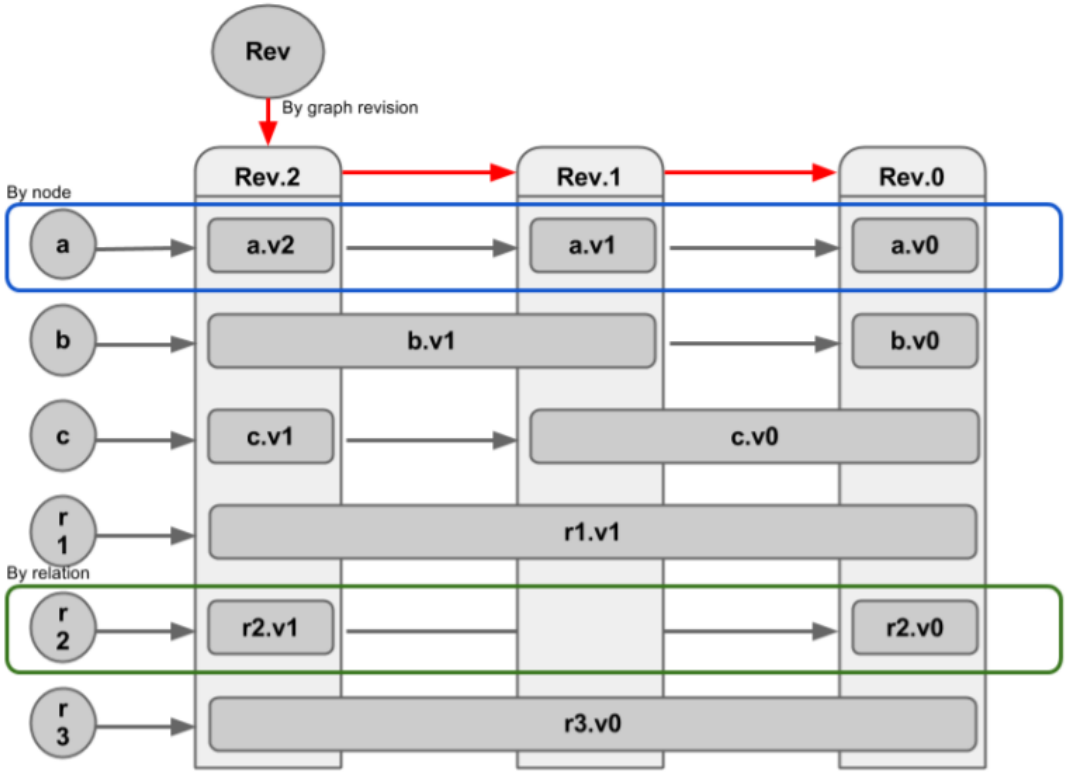

**9. ábra: Verzió gráf**

Az ábrán látható, hogy a legfrissebb verziójú entitást az eredeti nevén tároljuk, illetve kapcsolattal van hozzákötve egy régebbi verzió adata. Abban az esetben, ha további verziókat akarunk tárolni, egész egyszerűen tovább fűzzük a megfelelő entitás kapcsolati listáját.




**10. ábra: Csomópont módosítás**

Minden egyes változás valójában egy új állapotot eredményez a gráf modellünkben, amelyet a 11. ábrán láthatunk. Ezen látható a csomópont, a kapcsolat változás ábrázolása, az egyes állapotokkal együtt.




**11. ábra: Teljes verzió gráf**

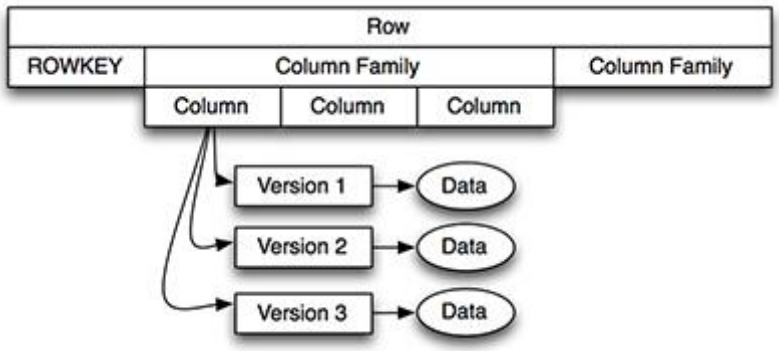
### 6.2. Oszlopalapú adatbázis

Az oszlopalapú adatbázisok esetében egy konkrét adatbázis, az HBase historizáció kezelését mutatom be. Egy oszloporientált adatbázis esetében is kifejezetten fontos lehet a különböző verziók nyomonkövetése. Épp emiatt, az oszlopalapú adatbázisok alapvetően támogatják az egyes verziók kezelését.

Az HBase egy rekordhoz vagy cellához szintén alapértelmezetten tárolja a verziót. Így minden módosítás után az adatbázis automatikusan tárolja a régebbi verziót és a kívánt adatot a megfelelő verzió kiválasztásával könnyű elérni. A verziószámmal ellátott értékeket csökkenő sorrendben tároljuk, hogy a legfrissebb érték maradjon legfelül, tehát a rekordok lekérésekor a legújabb verzió kerül mindig visszaadásra.

Annyi verziót készíthetünk, amennyit csak akarunk, de a verziók számát érdemes optimálisan eldönteni, mivel ez tárhelyfüggő. Ez azt jelenti, hogy minél több verziót tárolunk, annál több lemezterületet igényel.

Létrehozható egyéni verziókövető séma, ugyanakkor a felhasználók gyakran az alapértelmezett, a rendszer által generált megközelítést követik, amely a Unix idő alapján értékelődik ki.




**12. ábra: HBase 4-dimenziós adatmodell**

A 12. ábrán látható módon az HBase valójában egy négydimenziós adatmodellt határoz meg. A következő négy koordináta határozza meg az egyes cellákat:

- **Row Key**: Minden sorhoz egyedi row key tartozik. Nincs adattípusa és egy byte tömbként kezelik.
- **Column Family** (Oszlopcsalád): A soron belüli adatok oszlopcsaládokba vannak rendezve. Minden sorban ugyanazok az oszlopcsaládok vannak, de nem szükséges, hogy ezek egyforma oszlopminősítőket tartalmazzanak.

- **Column Qualifier vagy Column (Oszlopminősítő)**: Az oszlopcsaládok tényleges oszlopokat határoznak meg, amelyeket oszlopminősítőknek nevezünk. Az oszlopminősítőket tekinthetjük oszlopoknak.
- **Version (verzió)**: Minden oszlopnak konfigurálható számú verziója lehet és a fejlesztő hozzáférhet az oszlop kívánt verziójú adatához.

### 6.3. Dokumentumorientált adatbázis

Az egyik legelterjedtebb NoSQL adatbázisok közé tartoznak a dokumentumorientált adatbázisok. Ezek közül is a manapság leggyakrabban használt, legjobban támogatott a MongoDB. A MongoDB-t 2007-ben adták ki. Egy DoubleClick nevű csapatnak szüksége volt egy skálázható és flexibilis adatbázisra, mivel másodpercenként 400.000 reklámot szolgáltatott a rendszerük. Ez inspirálta őket arra, hogy tervezzenek egy olyan adatbázist, melyben minden adat JSON alapú dokumentumban van tárolva. A dokumentumok pedig gyűjteménybe (collection) vannak rendezve.

A dokumentumorientált adatbázisok kiválóan működnek akkor, ha nagyon sok adatot kérünk le és ezeket gyakran módosítanunk is kell. A legtöbb esetben azonban csak az adatok legfrissebb állapotára vagyunk kíváncsiak. Abban az esetben, ha a dokumentum egy korábbi állapota érdekel, akkor az adatbázisunkon alkalmazni kell valamilyen historizált tervezési mintát. Ilyen mintákra a MongoDB fejlesztői is nyújtanak példát. Ilyen minta a „Document Versioning Pattern” (DVP).

Ez a minta azt a problémát oldja meg, hogy bizonyos dokumentumok régebbi változatait a MongoDB-ben szeretnénk megtartani, ahelyett, hogy egy második tárolót, adatbázist vonnánk be a rendszerünkbe. Ennek érdekében minden dokumentumhoz hozzáadunk egy mezőt, amely lehetővé teszi, hogy a dokumentum verzióját nyomon tudjuk követni. Az adatbázisnak ezután két gyűjteménye lesz: az egyikben a legfrissebb adatok lesznek, amelyeket feltehetően a legtöbbször fog lekérni a felhasználó, a másikban pedig az adatok összes változata.

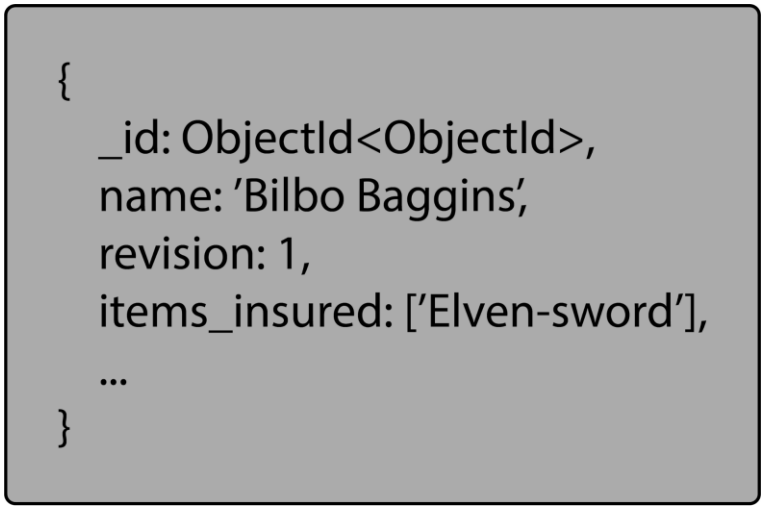
A DVP néhány dolgot feltételez az adatbázisban lévő adatokról és az alkalmazás adatelérési tulajdonságairól:

- A dokumentumoknak nincs lesz túl sok változata.
- Nem szükséges minden dokumentumot verzionálni.
- A legtöbb lekérdezés a dokumentum legfrisebb verzióján történik.

Abban az esetben, ha ezek a feltételek nem felelnek meg a használati esethez, akkor ez a minta nem biztos, hogy a legjobb megoldás a problémára. A MongoDB fejlesztői által számos minta van publikálva, amelyeket át lehet tanulmányozni a különböző használati esetekre vonatkozóan.

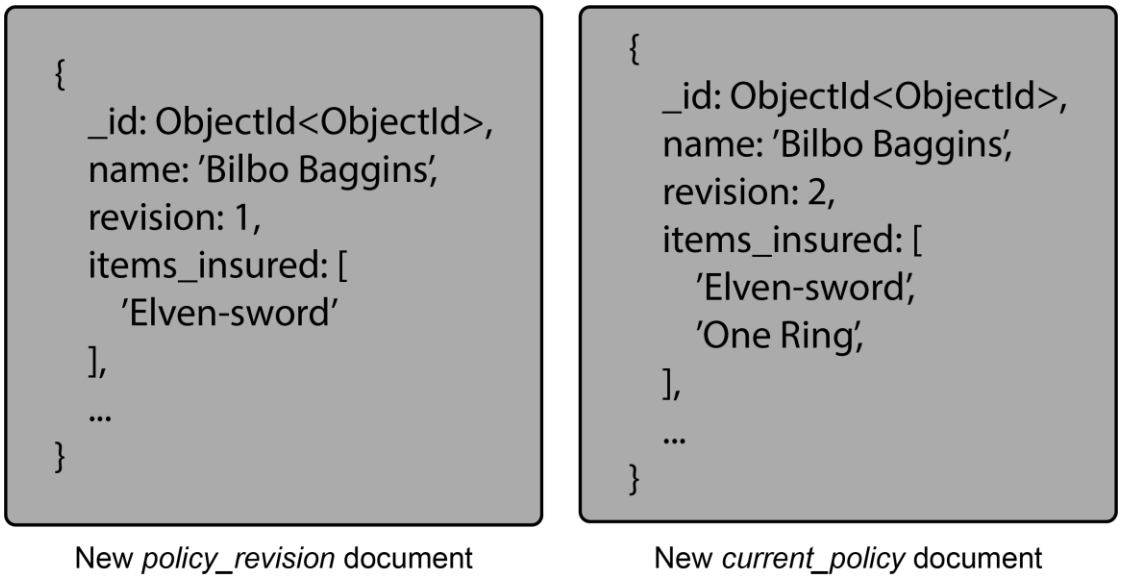
Tekintsük meg a következő nagyon leegyszerűsített példát a DVP alkalmazására vonatkozóan.

A dokumentumok ügyfeleket tárolnak, akik folyamatosan biztosításokat kötnek a saját eszközeire és szükségünk van arra, hogy az egyes szerződéskötéskor milyen eszközökre volt addig biztosítása. Így külön gyűjteményben tároljuk az ügyfeleket a legfrissebb biztosított eszközeikkel, ahogy a 13. ábrán látható.

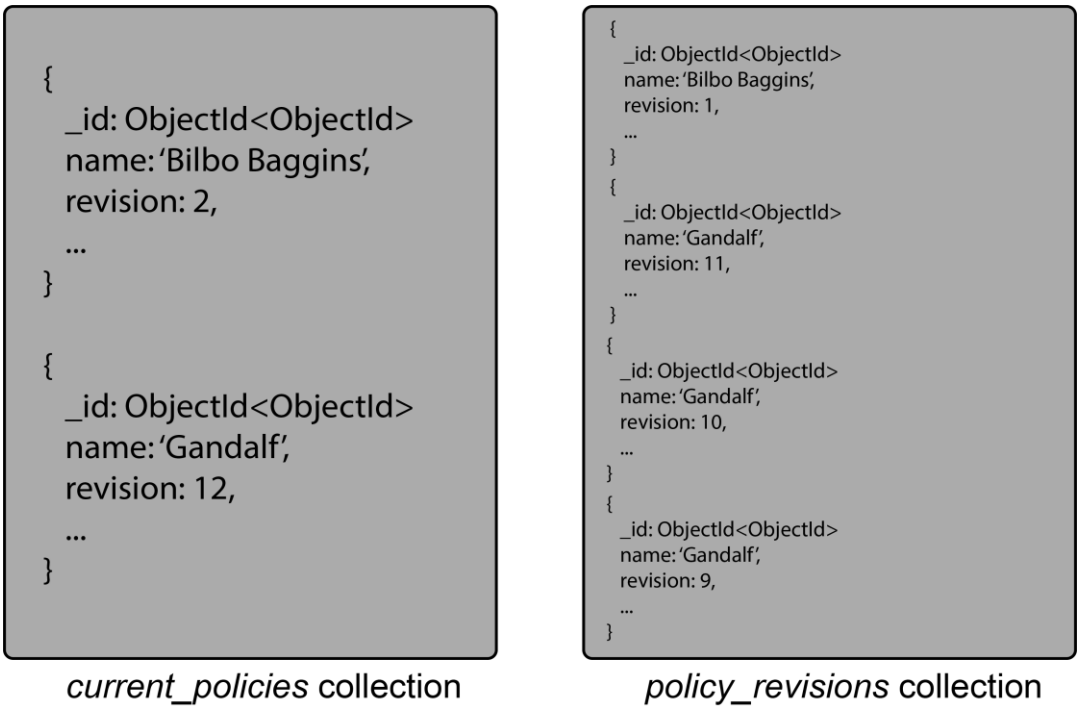



**13. ábra: Naprakész dokumentum**

Amennyiben módosítjuk az ügyfélhez tartozó biztosított eszközöket, akkor történik egy új dokumentum létrehozás a historizált adatokat tartalmazó gyűjteményben és egy módosítás a legfrissebb adatot tartalmazó gyűjteményben a megfelelő dokumentumon.




**14. ábra: Új és naprakészre módosított dokumentum**




**15. ábra: Document Versioning Pattern a MongoDB-ben**

Ha nyomon kell követni a dokumentumok változásait, a DVP alkalmazása erre egy kiváló lehetőség. Viszonylag könnyen megvalósítható és már meglévő dokumentumkészletre is alkalmazható. További előny, hogy az adatok legújabb verziójára vonatkozó lekérdezések továbbra is jól teljesítenek.

# 7. A megoldási módszer kiválasztása, a választás indoklása

## 7.1. Mobilalkalmazás

Az alapos irodalomkutatás során arra a megállapodásra jutottam az ügyféllel, hogy a mobilalkalmazás natív Android applikáció lesz, hiszen az egyesület tagjai a fák feltöltésére kifejezetten kapnak egy okostelefont kiváló minőségű kamerával. Így annak érdekében, hogy a lehető legstabilabb alkalmazás készüljön el számukra, Androidra natívan fogom lefejleszteni az appot a Kotlin nyelv előnyeit, egyszerűségeit kihasználva. Azon túl, hogy a Kotlin egyre népszerűbb nyelv, 2018 óta a Google hivatalosan a Kotlin nyelvet támogatja a natív Android fejlesztésre, így ennek köszönhetően is esett a döntésem a Kotlin nyelvre a Java helyett.

A mobilalkalmazás fejlesztése során kifejezetten nagy figyelmet fordítok a MAD (Modern Android Development) készségek és irányok betartására, ezentúl az architektúra tervezésénél az MVVM architekturális mintát fogom követni. Mindezt azért, hogy egy olyan minőségű alkalmazás készüljön el, ami hosszú távon könnyen karbantartható és átlátható minél több fejlesztő számára.

## 7.2. Webalkalmazás és API

A webalkalmazásokkal kapcsolatos kutatásaim során arra már hamar rájöttem, hogy mindenképp valamilyen keretrendszerben fog megtörténni a fejlesztés. A megfelelő keretrendszer kiválasztása addig okozott problémát, amíg meg nem találtam a Next.js-t.

A Next.js-nek köszönhetően nem kell két különböző keretrendszerben dolgoznom, hiszen nevezhetjük egy Full-Stack keretrendszernek, ami React-et használ FrontEnd oldalon és (Node.js) Express.js-t BackEnd oldalon. Van TypeScript támogatása, így a folyamatos típus biztonságnak köszönhetően jóval kevesebb bugot tartalmazhat a jövőben a kész alkalmazásom. Ezentúl automatikusan elvégez számos olyan optimalizációs feladatot, amelyet a fejlesztőnek kellene megcsinálnia, így sokkal hamarabb lehetséges elkészíteni az alkalmazást „production-ready” állapotra. Next.js keretrendszer alkalmazható úgy is, hogy csak a FrontEnd, vagy csak a BackEnd oldalát használom ki.

Ezek mind olyan előnyök, amelyek a jövőben, a skálázhatóság érdekében nagyon hasznosak lesznek. Ha Next.js-t csak FrontEnd oldalon használunk, már az is számos előnnyel jár. Ugyanakkor BackEnd oldalon ez nem így van és teljesen indokolatlan jelenleg Next.js keretrendszert használni a REST API fejlesztéséhez, hisz ennek az alkalmazásnak nem lesz felhasználói felülete.

Így az adminisztrációs rendszer (webalkalmazás) Next.js app lesz, míg a REST API Node.js-ben lesz programozva, az alapvető, minimalista Express.js-t használva.

### 7.3. Adatbázis

Az adatbázis szerkezete és felépítése alapvetően nem bonyolult ennél a rendszernél. Fákat, fatuskókat és felhasználókat (egyesület tagok) kell tárolni és a fák esetében szükséges a historizálás.

Viszonylag sok adatra számítunk és ezeket az adatokat gyakran fogják lekérni a telefonokon. Ezentúl fontos szempont a skálázhatóság, gyorsaság és a rugalmasság a sémában, hiszen nem minden fa tulajdonságához tartozik majd adat. Továbbá a kevés típusú egyed miatt nem alakul ki közöttük látványos kapcsolat.

Ezeket figyelembe véve úgy döntöttem, hogy egy NoSQL alapú és dokumentumorientált adatbázist, a MongoDB-t fogom használni a fejlesztés során. A döntésben nagy szerepet játszott az is, hogy a MongoDB fejlesztői közössége kifejezetten foglalkozik a historizálással és ehhez tervezési mintát is publikáltak.

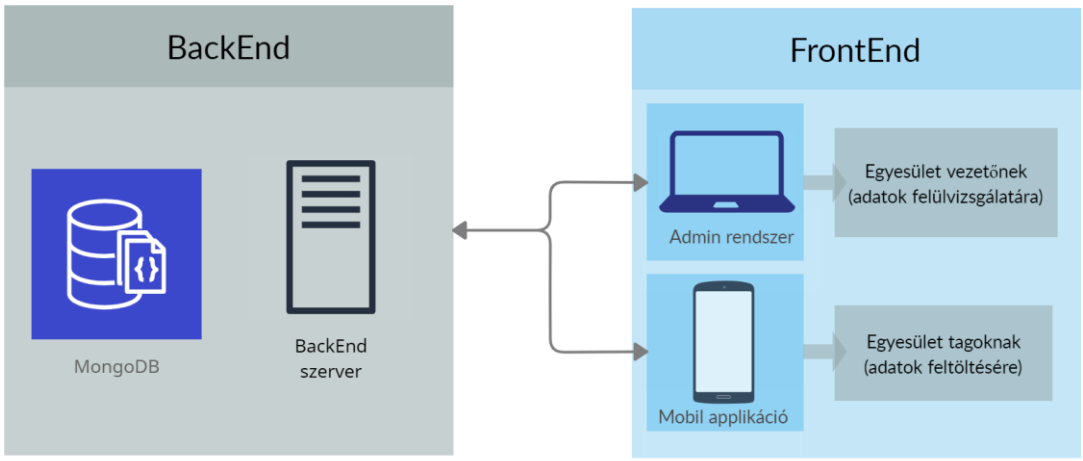
A tervezési mintájuk neve a Document Versioning Pattern, röviden DVP. Több feltételt meghatároztak, hogy mikor használatos a DVP. Az első ilyen feltétel az, hogy a dokumentumoknak nem lesz túl sok változata, ami esetünkben igaz, hiszen egy fával nem gyakran történik változás (pl.: betegség). A második feltételük az, hogy nem szükséges minden fát verzionálni, ami esetünkben szintén igaz, hiszen előfordulhat, hogy egy fával a kivágásáig nem történik semmi. A harmadik feltételük pedig az, hogy a lekérdezések nagy része a naprakész adatokon történjen, ami esetünkben újra igaz, hiszen a mobilalkalmazások csak a fák legújabb verzióit jelenítik meg és egyedül az adminisztrátor (egyesület vezető) az, aki a webalkalmazáson keresztül esetleg megnézi a fához tartozó korábbi adatot.

# 8. Rendszerterv

## 8.1. Áttekintő

A rendszer tervezése során egyértelművé vált, hogy a mobilalkalmazás egy csak frontend alkalmazás, tehát semmilyen szerver oldali kóddal nem fog rendelkezni. Ezentúl szükséges még az adminisztrációs rendszernek is egy felhasználó oldali, frontend megjelenés, továbbá szükséges még egy backend alkalmazás, amely kiszolgálja a frontendet. A Next.js-nek köszönhetően a webes frontend appot és a backend appot elkészíthetném egy Full-Stack Next.js alkalmazásként, viszont nem tartom jó megközelítésnek azt, hogy a teljes fakataszter rendszer csak félig full-stack, hiszen van egy mobilalkalmazás teljesen külön. Ugyanakkor megvan a jövőben a lehetőség, hogy ezt bármikor módosítani lehessen. A Next.js teljes mértékben támogatja azt a típusú fejlesztést, hogy a keretrendszernek csak a frontend vagy csak a backend lehetőségeit használjuk ki.

Ennek köszönhetően az alábbi módon terveztem meg a fakataszter rendszer alkalmazásait:




**16. ábra: Fakataszter rendszer**

## 8.2. BackEnd

### 8.2.1. Adatbázis - MongoDB

BackEnd oldalon természetesen megjelenik az adatbázis, amely tárolni fogja a naprakész és a historizált adatokat is.

Az adatbázist a MongoDB hivatalos oldalán, a https://www.mongodb.com/ oldalon fogom létrehozni a felhőben, mint egy „Database as a service”. Annak érdekében, hogy

ne kelljen folyamatosan böngészőn keresztül megnyitnom az adatbázist, telepítem a hivatalosan ajánlott asztali alkalmazást, a MongoDB Compass-t is.

#### 8.2.2. REST API – BackEnd szerver

A BackEnd alkalmazásunk REST API-ként fog funkcionálni, amely kiszolgálja a Next.js adminisztrációs rendszert és az Android mobilalkalmazást. Ezentúl kapcsolatban áll a MongoDB adatbázissal, amelyben adatmanipulációs és adatlekérdező típusú műveleteket hajt végre.

Node.js alkalmazás, a minimalista Express.js keretrendszerrel, amely tervezési mintáját tekintve egy rétegzett, JavaScript programozási nyelvben írt alkalmazás.

### 8.3. FrontEnd

#### 8.3.1. Android mobilalkalmazás

A mobilalkalmazásunk egy natív Android applikáció, amely Kotlin programozási nyelvben van írva, tervezési mintáját tekintve pedig MVVM-et alkalmaz. Az egyesület tagjai fogják használni adatok (fák, fatuskók) mentésére.

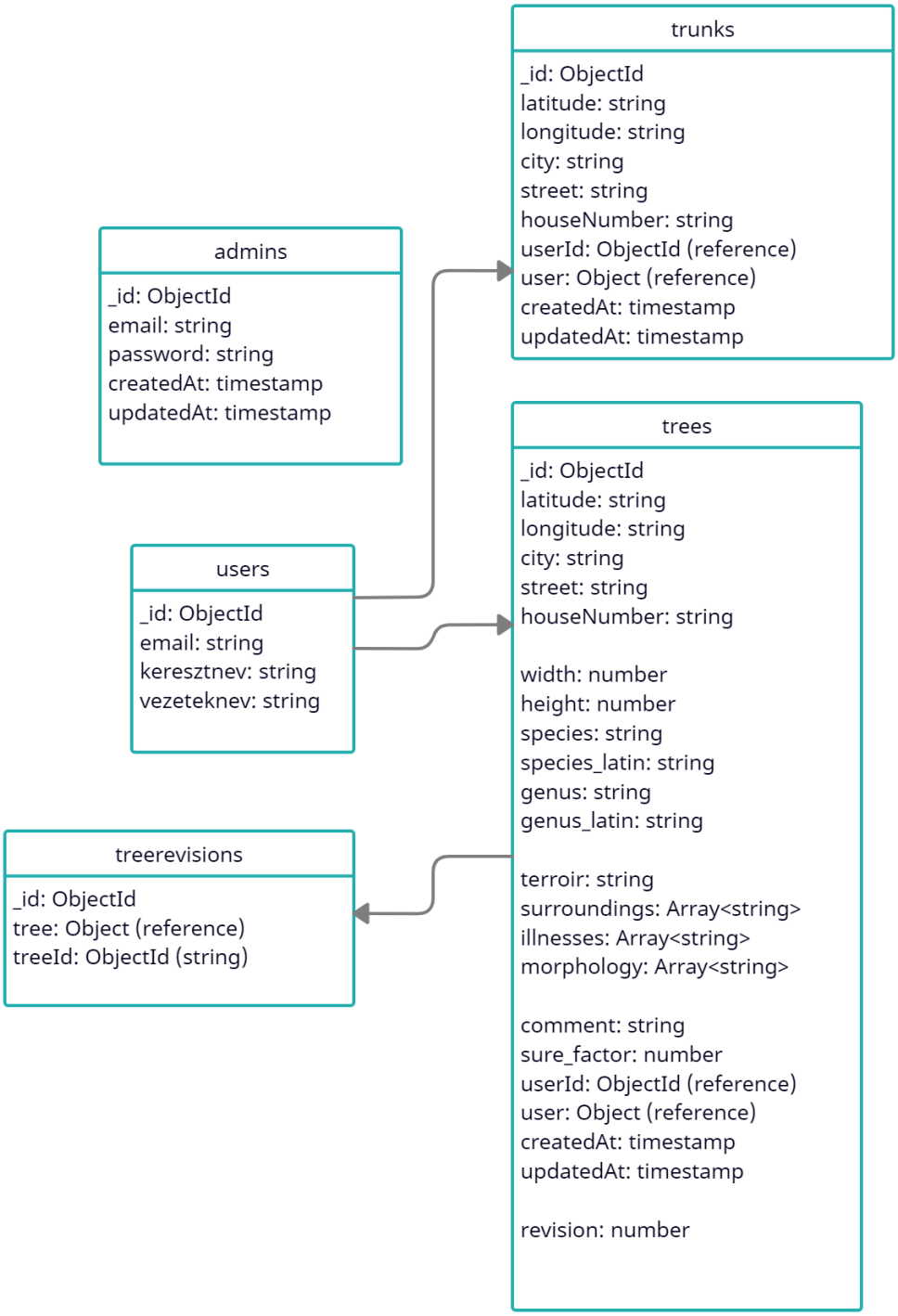
Kapcsolatban áll a REST API-val, hisz tőle kéri le a különböző adatokat (fa, fatuskó), amelyet térképen jelenít meg és a különböző mentési műveletek igényét továbbítja az API-nak.

#### 8.3.2. Webalkalmazás - Adminisztrációs rendszer

Az adminisztrációs rendszer pedig egy Next.js webalkalmazás lesz, amely az Android mobilalkalmazáshoz hasonlóan a REST API-nak küldi majd az adatlekérdező és adatmanipulációs kéréseket. Ezeket az adatokat pedig megjeleníti majd böngészőn keresztül az egyesület vezetőnek.

# 9. Implementáció

## 9.1. Adatbázis felépítése




**17. ábra: MongoDB Schema Design**

Az adatbázis séma a 17. ábrán látható.

Az egyes gyűjtemények és azok leírása:

1. „admins” gyűjtemény: Az adminisztrációs rendszer belépésére jogosult felhasználókat tárolja.

2. „users” gyűjtemény: Az egyesület tagjait tároló gyűjtemény. Ők jogosultak a mobilalkalmazás használatára.

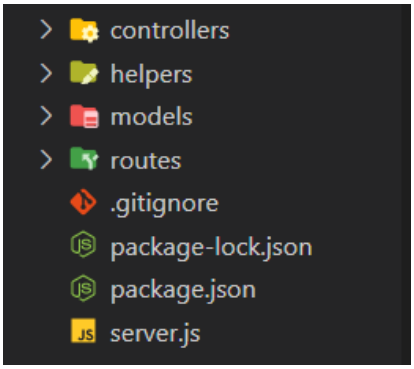
3. „trunks” gyűjtemény: A fatuskók adatait tároló gyűjtemény. Referenciát tartalmaz a létrehozó „egyesületi tag” felhasználóra vonatkozóan.

4. „trees” gyűjtemény: A fákról szóló legfrissebb adatokat tárolja. Ezeket az adatokat jelezte az egyesület, hogy a fakataszter rendszerben tárolni szeretné. Ilyen például a faj, nemzetség és ezek latin megfelelői. Továbbá vannak „tömb” adatszerkezetű mezők is, mint például a morfológia, vagy a betegség (egy fának több alaki jellegzetessége vagy betegsége lehet). Referenciát tartalmaz a létrehozó „egyesületi tag” felhasználóra vonatkozóan. Továbbá tartalmazza a DVP megvalósításához szükséges „revision” mezőt is.

5. „treerevisions” gyűjtemény: A fák historizált adatait tartalmazza. Minden módosításkor a fa jelenlegi állapota a „trees” gyűjteményből másolásra kerül a „treerevisions” gyűjteménybe és a „trees” gyűjteményben növekszik a revision mező értéke, valamint módosulnak a megfelelő adatok.

### 9.2. REST API fejlesztése

A fejlesztés kezdetekor először is megterveztem a rétegzett felépítését az alkalmazásnak, amely során a következőkre jutottam:

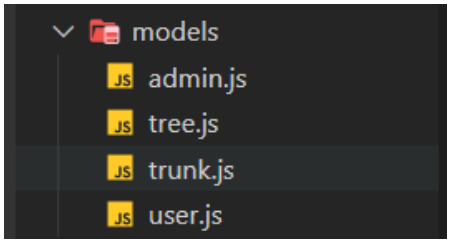



**18. ábra: REST API projekt struktúra**

A gyökérmappában helyezkedik el a szervert (alkalmazást) indító fájl és globális beállítások. Ezentúl látható a rétegzett felépítés: modell, útválasztás és controller.

### 9.2.1. Models

A modellek a 17. ábrának megfelelően vannak létrehozva a mongoose csomag segítségével. A mongoose a kódban leírt séma alapján automatikusan a beállított megszorításokkal és referenciával együtt létrehozza a gyűjteményeket és adatmanipuláció esetén a dokumentumot is megfelelően (létrehozás, módosítás) kezeli. Továbbá a mongoose segítségével nem szükséges MQL (MongoDB Query Language) nyelvben írni lekérdezéseket, helyette egyszerű metódusokat kell csak meghívni.




**19. ábra: REST API models**

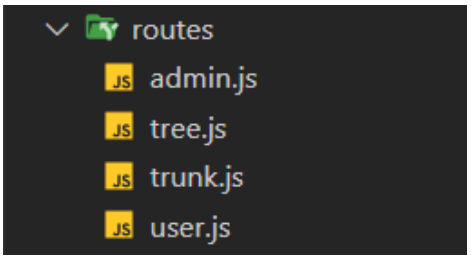
### 9.2.2. Routing

Az útválasztás megoldása a REST API alkalmazásoknál kifejezetten fontos. A frontend alkalmazások igényeit figyelembe véve az alábbi végpontokat hoztam létre:

- treeRoutes – Fa végpontok
  - tree/create: POST végpont– Fa létrehozás
  - tree/removeById: POST végpont – Fa törlése azonosító alapján
  - tree/getAll: GET végpont – Összes fa fontosabb adatainak lekérdezése
  - tree/getAllData: GET végpont – Összes fa adatainak lekérdezése
  - tree/getById: GET végpont – Egy fa összes adatának lekérdezése
  - tree/revisions/:id: GET végpont – Fa historizált adatainak lekérdezése
  - tree/update: PUT végpont – Fa módosítása

- trunkRoutes – Fatuskó végpontok
  - trunk/create: POST végpont – Fatuskó létrehozás
  - trunk/getAll: GET végpont – Összes fatuskó fontosabb adatait adja vissza
  - trunk/getAllData: GET végpont – Összes fatuskó adatai lekérdezése

- userRoutes – Felhasználó-kezelés végpontok
  - user/login: POST végpont – Felhasználó bejelentkezés
  - user/createUser: POST végpont – Felhasználó létrehozás
  - user/getById: GET végpont – Felhasználó lekérdezés ID alapján
  - user/getByTreeId: GET végpont – Lekérdezés fa ID alapján
  - user/getByTrunkId: GET végpont – Lekérdezés fatuskó ID alapján
  - user/deleteOne: POST végpont – Felhasználó törlés
  - user/getAll: GET végpont – Összes felhasználó lekérdezése

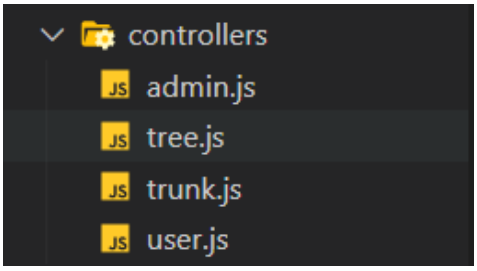
- 


**20. ábra: REST API routes**

A fa létrehozás végpont esetén be van állítva, hogy 3 képállományt is képes legyen fogadni.

### 9.2.3. Controllers

A megfelelő útvonalak továbbítják a megfelelő controllernek, hogy milyen „üzleti logikai" kód legyen végrehajtva. A következő ábrán látható controllereket hoztam létre:

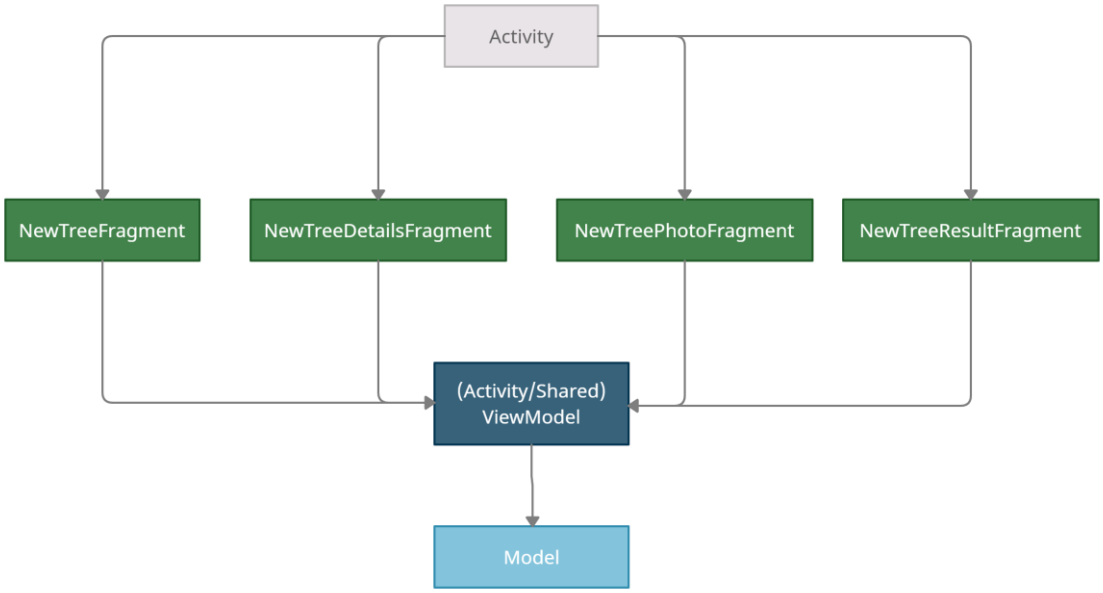



**21. ábra: REST API controllers**

Ezekben találhatóak azok a függvények, amelyek az egyes végpontok meghívása esetén lefutnak. Számos ilyen függvény van.

## 9.3. Android mobilalkalmazás fejlesztése

Az Android alkalmazás fejlesztése során betartottam az ajánlott, modern felépítést, ezentúl Kotlin nyelvet alkalmaztam, ugyanis 2018 óta hivatalosan ezt a nyelvet ajánlja a Google natív Android fejlesztésre. Összesen 1 Activity-t hoztam létre, és minden oldalnak 1-1 Fragment-et, így sokkal kevésbé erőforrásigényes az alkalmazás, ugyanis csak a Fragment-ek cseréje történik meg és nem egy új Activity jön létre, ha új lapra navigál a felhasználó. Ezentúl az MVVM architekturális mintának megfelelően ViewModel kapcsolatban áll a Fragment-el (View) és az üzleti logika a ViewModel-be van kiszervezve.

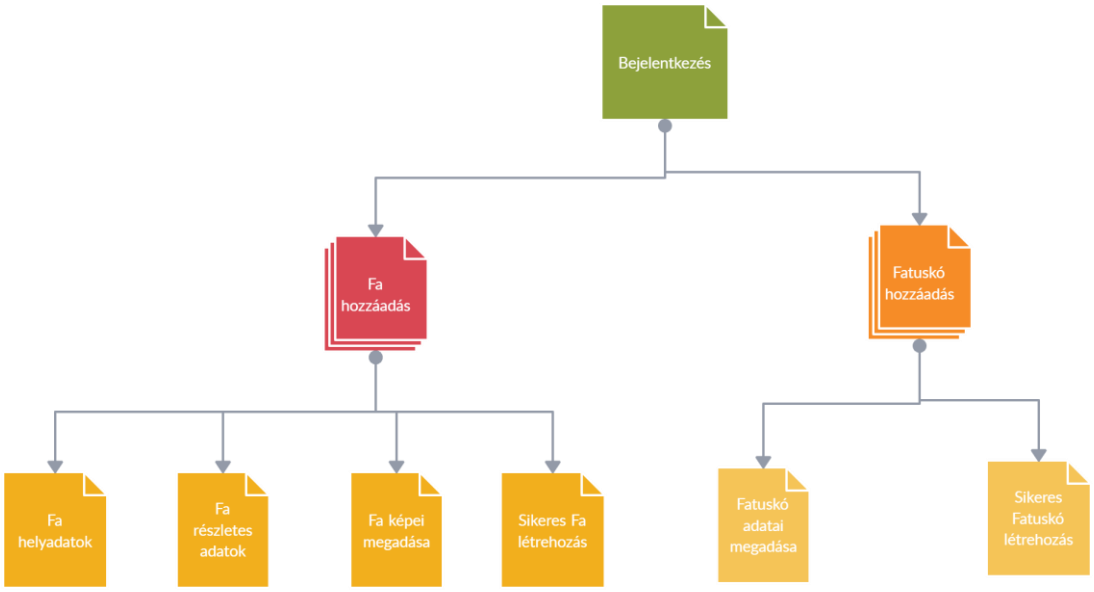



**22. ábra: Fakataszter mobil applikáció terve (fa létrehozás)**

Ahogy a 22. ábrán látható, ügyeltem arra, hogy egy rétegzett, jól karbantartható alkalmazást fejlesszek. A 22. ábrán csak a fához kapcsolódó Fragment és ViewModel látszódik. Természetesen ugyanezt a mintát alkalmaztam a fatuskó esetében is. A ViewModel egy úgynevezett „shared" vagy „activity" ViewModel, ami azt jelenti, hogy mindaddig, amíg az activity „életben van", azaz esetünkben nem lépünk ki az alkalmazásból, addig megtartja az adatokat. Ebben az esetben azért használom ezt a fajta ViewModel-t, mert a létrehozás több képernyőn keresztül történik és az adatokat „szállítani" kell egyikről a másikra. Így nem törlünk és hozunk létre mindig egy új ViewModel-t, hanem tulajdonképpen egy ViewModel példány van életben a létrehozás során.

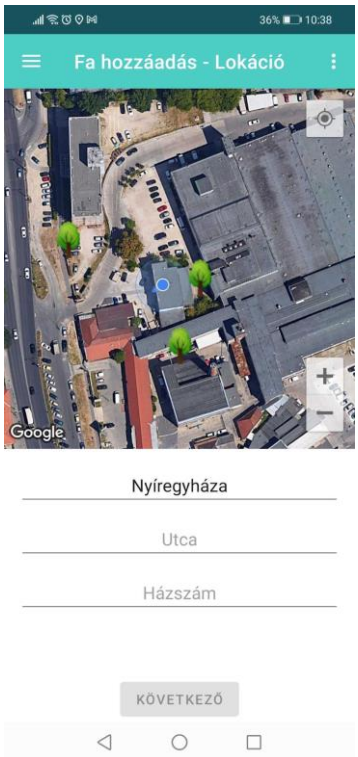
A ViewModel esetében még meg kell említeni, hogy a fa és fatuskó esetében is a LiveData (observable) segítségével kétirányú adatkötést hajtottam végre a felhasználói élmény javítása érdekében.

Ezentúl természetesen használok még további könyvtárakat, például a HTTP kérések küldésére. Ilyen a Retrofit, ami egy típusbiztos HTTP kliens Androidra.




**23. ábra: Fakataszter mobil applikáció site map diagram**

Mindezek alapján a 23. ábrán látható az elkészült alkalmazás oldaltérkép („Site Map”) diagrammja. Vannak olyan funkciók, amelyek nem külön oldalon, hanem a menüből érhetőek el. Ilyen például a térkép stílusának (műholdról alap stílusra váltás) módosítása, a kijelentkezés vagy a belépett felhasználó adatainak megjelenítése.



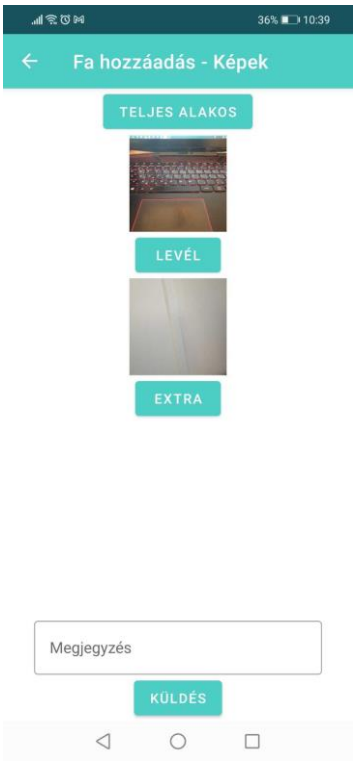

**24. ábra: Fa hozzáadás – Lokáció**

A 24. ábrán látható a fa hozzáadás első lépése. Ezen a képernyőn egy marker lehelyezése után automatikusan kitöltődnek a beviteli mezők, de ezek akár kézzel is módosíthatóak. Marker lehelyezése kötelező, ugyanis a latitude, longitude értékeket kötelező eltárolni az adatbázisban.


**25. ábra: Fa hozzáadás – Adatok**

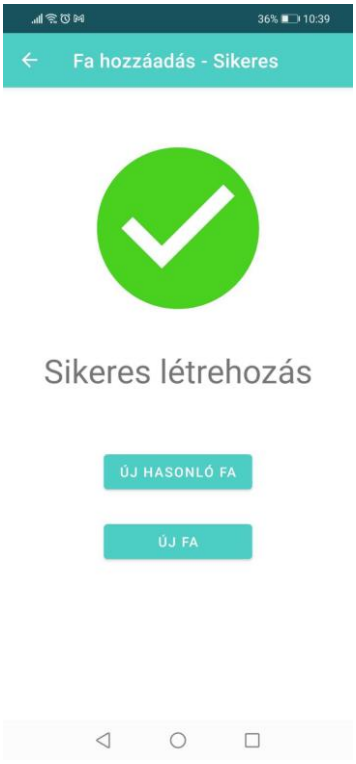
A 25. ábrán látható a második lépés. Itt olyan adatok megadása kötelező, mint például a nemzetség és faj, de a törzskerület és a magasság is.

A 26. ábrán látható a fa létrehozásának utolsó lépése. Ezen a képernyőn a „Teljes alakos”, „Levél” és „Extra” gombokra kattintva megnyílik a készülék kamerája és elkészíthető vele a fotó a fáról. A fotó elkészítése után a kép kisméretű változata megjelenik a felhasználó számára. A kép akárhányszor újra elkészíthető. Megjegyzés hozzáadása nem kötelező. A Küldés gomb akkor lesz aktív, ha elkészült a teljes alakos és a levélről készült kép. Az Extra kép készítése opcionális.




**26. ábra: Fa hozzáadás – Képek**

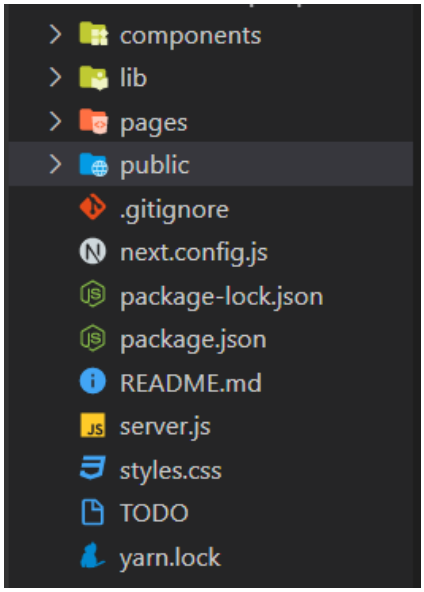
Sikeres létrehozás esetén az alkalmazás átirányít minket a 27. ábrán látható képernyőre. Itt eldönthetjük, hogy egy új hasonló fát adunk-e hozzá vagy egy teljesen új fát.




**27. ábra: Sikeres fa létrehozás képernyő**

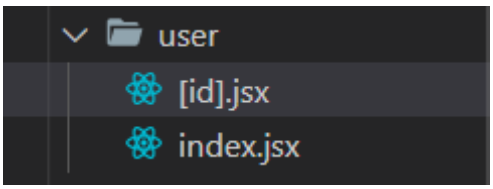
### 9.4. Adminisztrációs rendszer fejlesztése

Az adminisztrációs rendszer fejlesztése során erősen követtem a Next.js keretrendszer által ajánlott struktúrát.




**28. ábra: Adminisztrációs rendszer projekt struktúra**

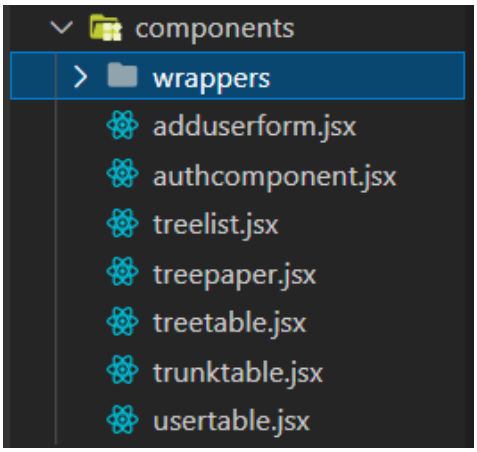
A pages mappában találhatóak meg a különböző lapok, amelyeket elérhet a felhasználó. Ezek „jsx” fájlok, amelyeket kifejezetten React-el együtt használnak. JSX segítségével egyszerűen köthetjük össze a megjelenítés logikáját a többi felhasználói felületi logikával, mint például az állapotok időbeni változása és események kezelése. Mindezt tulajdonképpen HTML nyelvhez hasonlító szintaxissal érjük el, azzal kiegészítve, hogy egyedi elemeket is létrehozhatunk és felhasználhatjuk már meglévő HTML elemekkel együtt.




**29. ábra: JSX fájlok**

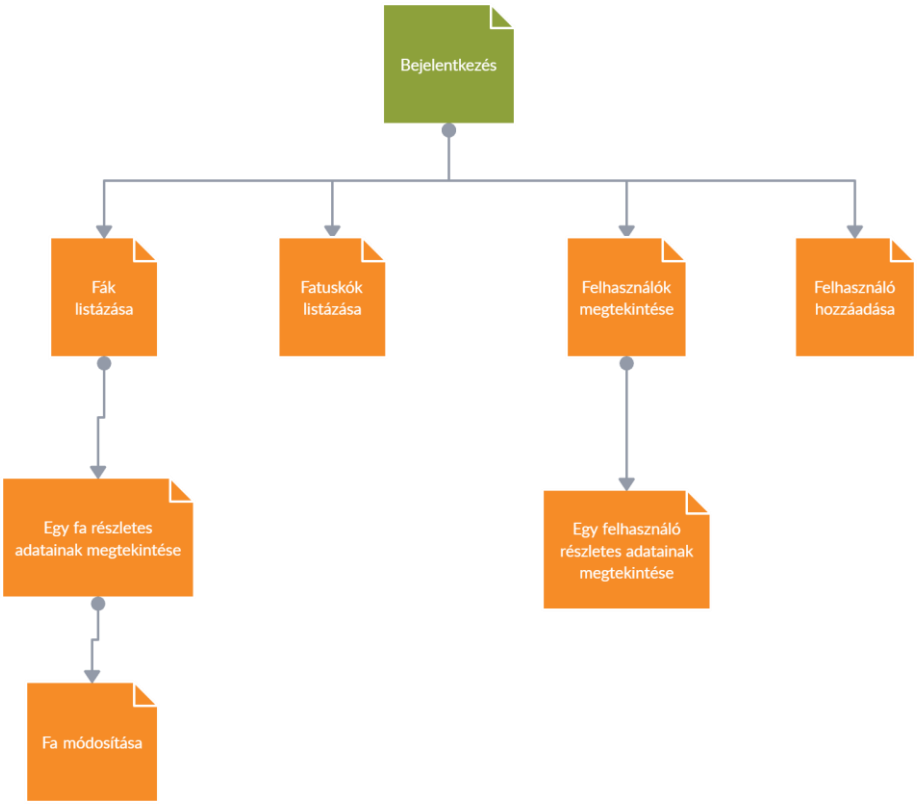
A 25. ábrán látható a Next.js routing rendszere is. A mappa neve az elérési útba épül be, az index.jsx fájl pedig az útvonal alapértelmezett oldalát jelenti. A szögletes zárójelekkel elnevezett oldal pedig egy dinamikus oldal, mely egy paramétert vár, amelyet felhasználhat az JSX oldal renderelésekor. Ennek megfelelően a „/user” útvonal az index.jsx által visszaadott oldalt (komponenst) jeleníti meg, a „/user/12551” pedig a 12551-es user adatait fogja megjeleníteni.

A „lib” mappában különböző kisegítő függvények találhatóak meg, a „components” mappában pedig a különböző kisegítő jsx komponensek. Ilyen például a táblázat, vagy a felhasználó létrehozását szolgáló űrlap. Ezeket a komponenseket a 26. ábra mutatja.




**30. ábra: Komponensek**

Mindezek alapján összefoglalva a 27-es ábrán látható az adminisztrációs rendszer site map diagramja.




**31. ábra: Adminisztrációs rendszer site map diagram**

Ez alapján tekintsük meg a fontosabb funkciókat tartalmazó képernyőképeket. Az első ilyen a fák listázása, amely a 32. ábrán látható. A táblázat viszonylag széles. Megjelenik még rajta a fát létrehozó felhasználó neve is, rajta hivatkozással a felhasználó részletes adatai oldalra. A fának a részletes adatait a további adatok oszlopban lévő azonosítóra kattintva érjük el.

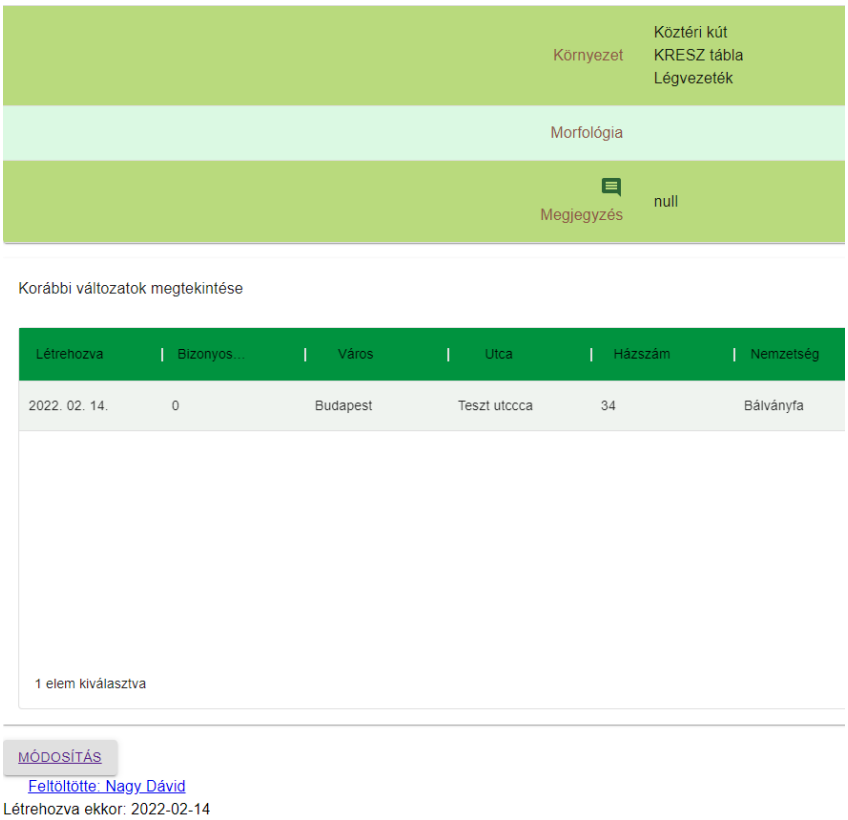
Fák megtekintése / Fatuskók megtekintése / Felhasználók megtekintése / Felhasználó hozzáadása / Kijelentkezés

| További adatok | Város | Utca | Házszám | Nemzetség | Faj | Nemzetség (l... |
|---|---|---|---|---|---|---|
| 1 | Budapest | Bécsi út | 267 | Lepényfa | Tövises | Gleditsia |
| 2 | Budapest | Bécsi út | 2 | Lepényfa | Tövises | Gleditsia |
| 3 | Budapest | Farkastorki út | 21-23 | Lepényfa | Tövises | Gleditsia |
| 4 | Budapest | Bécsi út | 267 | Gyertyánszil | Egyéb | Zelkova |
| 5 | Budapest | Vörösvári út | 133 | Berkenye | Lisztes | Sorbus |
| 6 | Budapest | Bécsi út | 267 | Bálványfa | Mirigyes | Ailanthus |
| 7 | Budapest | Bécsi út | 267 | Berkenye | Lisztes | Sorbus |
| 8 | Budapest | Bécsi út | 269 | Júdásfa | Közönséges | Cercis |
| 9 | TestCity | TestStreet | 10/b | testGenus | testKind | testGenusLatin |
| 10 | TestCity | TestStreet | 10/b | testGenus | testKind | testGenusLatin |

**32. ábra: Fák listázása**

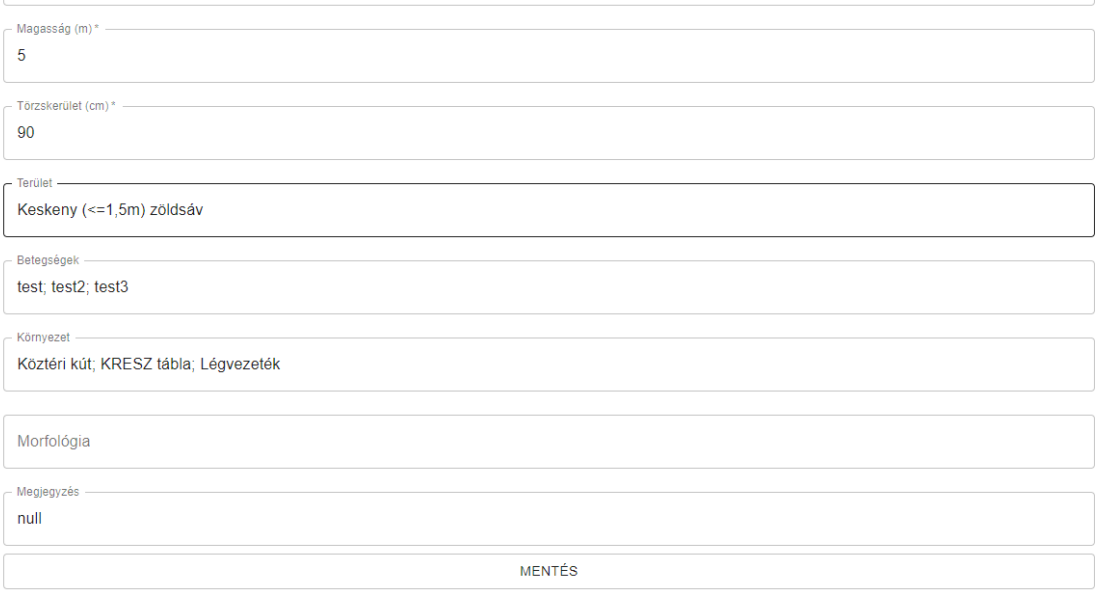
A fa részletes adatai oldalon megjelenik a fáról minden adat. Ezentúl betöltődnek az elkészült képek is. Szintén ugyanezen az oldalon lehetséges a fát módosítás gombra rákattintani és megtekinteni a korábbi verziókat is.


**33. ábra: Fa részletes adatai**




**34. ábra: Korábbi fa verziók megtekintése**

A fa módosítása képernyő egy része a 34. ábrán látható. Ezen az oldalon betölt minden beviteli mező a jelenlegi értékkel és ezeket átírva, majd a mentés gombra kattintva történik meg a módosítás. Azoknál a beviteli mezőknél, ahol több értéket is tárolunk (tömb), ott pontosvesszővel kell elválasztani az egyes értékeket.




**35. ábra: Fa módosítása**

# 10. Tesztelés és eredmény

Az alkalmazás fejlesztésének egy olyan szakaszában, amikor már voltak használható funkciók, elkezdődött a felhasználói tesztelés. Előtte az egyesületnek bemutattam a mobilalkalmazást, a vezetőnek pedig az adminisztrációs rendszert is. Mivel az egyesületnek viszonylag gyorsan szüksége volt az alkalmazásra, így részükről is olyan igény jött, hogy minél hamarabb tudják kipróbálni és ne menjen az idő sokáig fejlesztéssel. Ennek megfelelően nem készültek unit tesztek, amik körülbelül másfélszeresére növelték volna a fejlesztési időt, helyette a felhasználói tesztelés kezdődött el.

Az alkalmazásokat természetesen én magam is teszteltem, így a környezet részletes leírását és eredményeit is ismertetem a két alkalmazás esetében a továbbiakban.

## 10.1. Mobilalkalmazás tesztelése és eredménye

A mobilalkalmazást saját telefonomon teszteltem.

A tesztelés során a saját okostelefonom specifikációját az 1. táblázat tartalmazza.

| | |
|---|---|
| **Eszköz neve** | HUAWEI Mate 20 lite |
| **Modell** | SNE-LX1 |
| **Android verziója** | 10 |
| **Processzor** | Hisilicon Kirin 710 |
| **RAM** | 4 GB |

**1. táblázat: Mobilalkalmazást tesztelő eszköz leírása**

Minden követelmény, amely szerepelt a specifikációban, tesztelésre került és ezen tesztesetek eredményét a 2. táblázat szemléleteti.

| | |
|---|---|
| **Bejelentkezés** | sikeres |
| **Kijelentkezés** | sikeres |
| **Fák megjelenítése térképen** | sikeres |
| **Fatuskók megjelenítése térképen** | sikeres |
| **Térkép megjelenítésekor GPS alapján a** | sikeres |

| | |
|---|---|
| **felhasználó helyére pozícionálás** | |
| **Fatuskó adatainak megadása** | sikeres |
| **Új fatuskó mentése** | sikeres |
| **Fa adatainak megadása képekkel együtt** | sikeres |
| **Új fa mentése** | sikeres |
| **Új „hasonló fa” mentése** | sikeres |

**2. táblázat: Mobilalkalmazás tesztek eredményei**

A saját tesztelésem után az applikációt publikáltam a Google Play Store áruházba, hogy megfeleljek az ügyfélnek és minél hamarabb elkezdhessék az egyesületi tagok a tesztelést.

A felhasználók kifejezetten a mobilalkalmazást tesztelték több héten keresztül, több típusú és márkájú okostelefonnal. Az alkalmazást Google Play Store áruházból töltötték le és telepítették a telefonjukra. Számos kisebb felhasználói élményt javító visszajelzést kaptam, amelyeket a lehető leghamarabb javítottam (pl.: vissza gomb lenyomását kellett felülírni), de előfordultak valódi „bugok” is, amelyek javítása szintén a lehető leggyorsabban történt meg.

A felhasználói tesztelés azért is volt eredményes, mert olyan módosítani való funkciókat, amelyek a unit teszten átmentek volna – hiszen nem a funkcióval volt a hiba –, azok csak akkor derülnek ki, ha a felhasználó a kezébe fogja a telefont és az éles helyzetnek megfelelő környezetben teszteli. Ezekkel a visszajelzésekkel bár elsősorban a felhasználói élmény lett a legjobban javítva, de eredményes feltárás és javítása is megtörtént néhány működésbeli hibának is.

### 10.2. Webalkalmazás tesztelése és eredménye

A webalkalmazást teszteltem lokálisan a saját számítógépemen, amely Windows operációs rendszerrel rendelkezik. Az alkalmazás böngészőben fut, így független attól, hogy milyen eszközről nyitjuk meg. Lehet Windows/macOS számítógép, Android vagy iOS telefon. Az a fontos, hogy rendelkezzen a készülék olyan böngészővel, amely még támogatott.

Minden követelmény, amely szerepelt a specifikációban, tesztelésre került és ezen tesztesetek eredményét a 3. táblázat mutatja meg.

| | |
|---|---|
| **Bejelentkezés** | sikeres |
| **Kijelentkezés** | sikeres |
| **Fák megjelenítése táblázatban** | sikeres |
| **Fatuskók megjelenítése táblázatban** | sikeres |
| **Felhasználók (egyesületi tagok) megtekintése** | sikeres |
| **Felhasználó (egyesületi tag) hozzáadása** | sikeres |
| **Felhasználó (egyesületi tag) törlése** | sikeres |
| **Egy fa részletes adatainak megtekintése** | sikeres |
| **Fa módosítása historizálással** | sikeres |
| **Fa historizált adatainak megtekintése** | sikeres |

**3. táblázat: Adminisztrációs rendszer tesztek eredményei**

Miközben zajlott a mobilalkalmazás tesztelése az egyesületi tagok által, ugyanebben az időben az egyesület vezetője elkezdte aktívan használni az adminisztrációs rendszert. Itt már tényleg csak 1-2 felhasználói élményt javító módosítást kellett elvégezni a visszajelzés miatt, hiszen ennél a rendszernél nem a megjelenés, hanem az adatok megtekintése és a funkciók működése a legfontosabb.

# 11. Továbbfejlesztési lehetőségek

A Fakataszter rendszer továbbfejlesztése még több szempontból is lehetséges és véleményem szerint nagyon hasznos lehetőségeket rejt még. A rendszer továbbfejlesztése lehetséges még fejlesztői szempontból és funkcionalitás szempontból is.

Fejlesztői szempontból a jelenlegi kódbázist, úgy gondolom, hogy a következőképpen lehet még továbbfejleszteni:

- TypeScript alkalmazása BackEnd-en
- TypeScript alkalmazása az adminisztrációs rendszer esetében
- Képek tárolása a lokális szerver helyett egy külső tárhelyen (pl.: Google Cloud, Amazon)
- E2E tesztelés mobilon és webes rendszerben is

Funkcionalitás szempontjából sokkal több és érdekesebb ötletem van a továbbfejlesztést illetően:

- Mobilalkalmazásban meg lehessen tekinteni bárkinek, autentikáció nélkül térképen a fákat. (Fák olvashatóvá/megtekinthetővé tétele bárkinek)
- Mobilalkalmazásban is lehessen módosítani a fát historizálva
- Módosítás esetén historizálás opciójának ki- és bekapcsolása
- Egyesületi tagok regisztrálhatnak, de csak az egyesület vezető hagyhatja jóvá a regisztrációt, így nem az adminisztrátornak kell felvinni az adatokat
- Térképes megtekintés a fákról és fatuskókról az adminisztrációs rendszerben is
- Reporting funkciók a webes rendszerbe

# 12. Összefoglalás

A szakdolgozatom témáját a jelenlegi cégemnél rám bízott projekt adta. Az első meeting során bemutatkozott az ügyfél és az egyesület ismertetésre került, amivel tisztázódtak a szerepkörök. Ezután átbeszéltük azt, hogy mire van szükségük. Részletesen kitértünk a különböző igényekre. A vezető viszonylag magabiztosan tudta azt, hogy mire akarja használni a fakataszter rendszert és mik a konkrét igényei, viszont a „hogyan" kérdésben erősen kikérte a véleményemet. Megbeszéltük, hogy az egyesületi tagok a legegyszerűbben egy mobilalkalmazáson keresztül tudják lefotózni a fákat és feltölteni annak adatait, a vezető pedig egy webalkalmazáson keresztül tudja ezt a legjobban monitorozni és módosítani azt. Kitértünk arra is, hogy az egyesületnek rendelkezésére állnak Android eszközök annak érdekében, hogy a feltöltött fák képeinek minősége nagyjából azonos legyen. Miután mindezt meghatároztuk, elkezdődött az irodalomkutatás, majd a fejlesztés.

A folyamatot irodalomkutatással kezdtem. Először felkutattam, hogy az Android fejlesztésnek hogyan kezdjek neki. A különböző dokumentációk olvasásán túl, számos oktatóvideót néztem meg híres programozóktól. Még egy online kurzuson is részt vettem a minőségi tanulás érdekében. Megismertem a legmodernebb eszközöket, keretrendszereket mind weben, mind pedig Androidon és ezek alapján választottam ki számomra a legideálisabbat. A tanulás és irodalomkutatás után elkezdődött az alkalmazások leprogramozása.

A fejlesztés kezdetén elkészítettem a szükséges diagramokat, terveket (wireframe), amelyeket a megrendelővel egy megbeszélés során validáltam. Ezt követően elkezdődött a tényleges programozás.

A fejlesztés során nagy segítségemre volt, hogy szinte folyamatosan rendelkezésemre állt az egyesület vezetője, így szinte bármikor tudtam tőle kérdezni, ha elbizonytalanodtam. Ez nagyban könnyítette és gyorsította a folyamatot. Számos nehézséggel és problémával találkoztam a munka során, amelyeket előbb-utóbb sikerült megoldanom. Ilyen volt például a több kép feltöltése telefonról a webes rendszerbe vagy elkészült képek tömörítése és megjelenítése kisebb méretben a mobilappon belül.

Az elkészült fakataszter rendszer jelenleg még csak demó stádiumban van. Folyamatosan megy a belső tesztelés és hamarosan nagyobb felhasználóközösség is kialakul majd. Úgy gondolom, hogy rengeteg új tapasztalatot és tudást szereztem ebben a projektben, amelyeket hasznosítani fogok a jövőben.

## 13. Summary

The topic of my thesis was a project assigned to me at my current company. During the first meeting, the client was introduced and the association was introduced, which clarified the roles. We then talked through what they needed. We went into detail about the different needs. The manager was relatively confident in what he wanted to use the tree cadastre system for and what his specific needs were, but he asked me for my opinion on the "how". We discussed that the easiest way for the association members to photograph the trees and upload their data was through a mobile app, and the manager could best monitor and modify it through a web app. It was also pointed out that the association has Android devices at its disposal to ensure that the quality of the images of the uploaded trees is roughly the same. Once all this was determined, the literature research and then the development began.

I started the process with a literature research. First, I researched how to get started with Android development. Besides reading various documentation, I watched several tutorial videos from famous programmers. I even took an online course for quality learning. I learned about the latest tools and frameworks both on the web and Android and based on these, I chose the most ideal one for me. After studying and researching the literature, I started programming the applications.

At the beginning of the development, I prepared the necessary diagrams and designs (wireframe), which I validated in a meeting with the client. After that, the actual programming started.

During the development, it was a great help that the head of the association was available to me almost all the time, so that I could ask him almost at any time if I was unsure. This made the process much easier and faster. I encountered many difficulties and problems during the work, which I managed to solve sooner or later. For example, uploading multiple images from a phone to the web system or compressing and displaying completed images in a smaller size within the mobile app.

The completed tree cadastre system is currently only in the demo stage. Internal testing is ongoing and a larger user community will be established soon. I think I have gained a lot of new experience and knowledge from this project, which I will use in the future.

# 14. Irodalomjegyzék


[1] Dendrocomplex: Mire jó a fakataszter?,
(https://dendrocomplex.hu/hir/mire-jo-a-fakataszter),
utoljára megtekintve: 2021.11.29.

[2] Tanács Attila, Kálmán Kornél: Android fejlesztés,
(https://www.inf.u-szeged.hu/~tanacs/oktatas/szggraftg15/Android_fejlesztes.pdf),
utoljára megtekintve: 2021.11.29.

[3] Rédecsi Máté, Tóth Gábor: Android architektúra
(http://nyelvek.inf.elte.hu/leirasok/Android/index.php?chapter=1),
utoljára megtekintve: 2021.11.29.

[4] Sicz-Mesziár János, Mezei József: Android alkalmazásfejlesztés,
(https://users.nik.uni-obuda.hu/malk/android/ea_2017_osz/03_-_Android_komponensek_activity_fragment.pdf),
utoljára megtekintve: 2021.11.29.

[5] Sergey L.: Android Jetpack Library Manual,
(https://www.cleveroad.com/blog/android-jetpack-library-a-robust-tool-to-make-development-faster),
utoljára megtekintve: 2021.11.29.

[6] Newbedev: MVVM,
(https://newbedev.com/when-using-mvvm-on-android-should-each-activity-have-one-and-only-one-viewmodel),
utoljára megtekintve: 2021.11.29.

[7] Horváth Győző, Tarcsi Ádám: Keretrendszerek, tervezési minták webes alkalmazásokban,
(http://ade.web.elte.hu/wabp/lecke7_lap1.html),
utoljára megtekintve: 2021.11.29.

[8] Lalithnaryan C.: Node.js versus Next.js - A React Approach,
(https://www.section.io/engineering-education/node-versus-next-react-approach/),
utoljára megtekintve: 2021.11.29.

[9] Node.js - Express Framework,
(https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm),
utoljára megtekintve: 2021.11.29.

[10] Most Popular Backend Frameworks,
(https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2021/),
utoljára megtekintve: 2021.11.29.

[11] Dawid Karczewski: Best Frontend Frameworks in 2021,
(https://www.ideamotive.co/blog/best-frontend-frameworks),
utoljára megtekintve: 2021.11.29.

[12] Batista Tone: Why NextJs and How to start ?, (https://medium.com/swlh/why-nextjs-and-how-to-start-6bf5c28f5a87), utoljára megtekintve: 2021.11.29.

[13] Anders Nawroth: Social Networks in the Database: Graph Database, (https://neo4j.com/blog/social-networks-in-the-database-using-a-graph-database/), utoljára megtekintve: 2021.11.29.

[14] Arnaud Castelltort, Anne Laurent: Representing history in graph-oriented nosql databases: A versioning system. *HAL*, 2019

[15] Shashwat Shriparv: Learning HBase. *Packt*, 2014

[16] Introduction to HBase, the NoSQL Database for Hadoop, (https://www.informit.com/articles/article.aspx?p=2253412), utoljára megtekintve: 2021.11.29.

[17] MongoDB, (https://www.mongodb.com/), utoljára megtekintve: 2021.11.30.

[18] Daniel Coupal, Ken W. Alger: Building with Patterns: The Document Versioning Pattern, (https://www.mongodb.com/blog/post/building-with-patterns-the-document-versioning-pattern), utoljára megtekintve: 2021.11.30.

# 15. Ábrajegyzék

# 16. Táblázatjegyzék